WicePlus

C & Assembly

For EM78 Series Microcontrollers

USER'S MANUAL

Doc. Version 2.9

ELAN MICROELECTRONICS CORP.

Nov. 2008

Trademark Acknowledgments

IBM is a registered trademark and PS/2 is a trademark of IBM.
Windows is a trademark of Microsoft Corporation.

ELAN and ELAN logo are trademarks of ELAN Microelectronics Corporation.

Copyright © 2008 by ELAN Microelectronics Corporation
All Rights Reserved
Printed in Taiwan

The contents of this User's Guide (publication) are subject to change without further notice. ELAN Microelectronics assumes no responsibility concerning the accuracy, adequacy, or completeness of this publication. ELAN Microelectronics makes no commitment to update, or to keep current the information and material contained in this publication. Such information and material may change to conform to each confirmed order.

In no event shall ELAN Microelectronics be made responsible for any claims attributed to errors, omissions, or other inaccuracies in the information or material contained in this publication. ELAN Microelectronics shall not be liable for direct, indirect, special incidental, or consequential damages arising from the use of such information or material.

The software (WicePlus) described in this publication is furnished under a license or nondisclosure agreement, and may be used or copied only in accordance with the terms of such agreement.

ELAN Microelectronics products are not intended for use in life support appliances, devices, or systems. Use of ELAN Microelectronics product in such applications is not supported and is prohibited.
NO PART OF THIS PUBLICATION MAY BE REPRODUCED OR TRANSMITTED IN ANY FORM OR BY ANY MEANS WITHOUT THE EXPRESSED WRITTEN PERMISSION OF ELAN MICROELECTRONICS.

TCC2 (Tiny C Compiler version 2) distribution note

2008/07/10

From now on we dogmatically call C Compiler which has been embedded in WicePlus for long time as TCC1 (Tiny C Compiler version 1 or ELAN Tiny C Compiler version 1).
TCC2 (Tiny C Compiler version 2 or ELAN Tiny C Compiler version 2) is based on the TCC1 and improve it. There are some difference between TCC1 and TCC2. For example TCC2 supports EM78P510N but TCC1 not; TCC2 supports TBRD instruction but TCC1 not. The Table below shows them. For some reason we reserve TCC1 and TCC2 in the following distribution of WicePlus. Actually users can just modify somewhere such as important register save and back up, C system save and back up registers in interrupt function and notification reserve register (0x10) in projects between TCC1 and TCC2 to develop their project. But we don't suggest that.

Difference between TCC1 and TCC2

	TCC1	TCC2
Reserve common register for Compiler	Non	0x10 *
C system back up	manual	Auto *
interrupt function	interrupt service functions scatter on corresponding subroutine	All interrupt service coding in one function and using global varible IntVecIdx occupied 0x10 to know interrupt source. So we suggest users not to use 0x10 *
Size 255 Constant array number in a program page	Max. 2	Max. 4 **
EM78P510N	unavailable	Available
eKTP8733N	unavailable	Available
EM78P220N	unavailable	Available
EM78P210N	unavailable	Available
TBRD instruction	unavailable	available

*: Please reference to 7.11

**: Depend on IC feature. Some IC just can contain one. Please reference to IC's spec.

If you have any problems, please contact myjian@emc.com.tw. We will help you to solve your problems as soon as possible.

Table of Contents

Chapter 1 Introduction 10

1.1 Overview 10

1.2 System Requirements 10

1.2.1 Host Computer 10

1.3 Software Installation 11

1.4 ANSI Compatibility 11

Chapter 2 WicePlus Interface 12

2.1 Overview 12

2.2 WicePlus Sub-Windows 13

2.2.1 The “Project” Window 13

2.2.2 The “Editor” Window 16

2.2.3 The “Special Register” Window 22

2.2.4 The “CallStack” Window 24

2.2.5 The “General Registers (Ram Bank)” Window 26

2.2.6 The “Watch” Window 28

2.2.7 “Data RAM” Window 34

2.2.8 “LCD RAM” Window 35

2.2.9 “Output” Window 39

2.3 WicePlus Menu Bar and its Commands 41

2.3.1 File Menu 41

2.3.2 Edit Menu 42

2.3.3 View Menu 46

2.3.4 Project Menu 47

2.3.5 Debug Menu 49

2.3.6 Tool Menu 59

2.3.7 Option Menu 64

2.3.8 IDE Menu 73

2.3.9 Window Menu 73

2.3.10 Help Menu 73

2.4 Toolbar 75

2.4.1 Toolbar Icons and Functions 75

2.5 Document Bar 77
2.6 Status Bar 77
2.7 Hot Key 78

Chapter 3 Getting Started 79

3.1 Hardware Power-up 79
3.2 Starting the WicePlus Program 79
3.2.1 Connect Dialog 79
3.2.2 Code Option Dialog 81
3.2.3 Accelerate Reading Registers Dialog 81
3.3 Create a New Project 81
3.4 Add and Remove Source Files from/to Project 83
3.4.1 Create and Add a New Source File for the Project 83
3.4.2 Add Existing Source Files to the New Project 84
3.4.3 Deleting Source Files from Project 85
3.5 Editing Source Files from Folder/Project 85
3.5.1 Open Source File from Folder for Editing 85
3.5.2 Open Source File from Project for Editing 86
3.6 Compile the Project 86
3.7 Dumping the Compiled Program to ICE 88
3.8 Debugging a Project 88
3.8.1 Breakpoints Setting 91

Chapter 4 Assemble and link 92

4.1 Assembler and linker process flow 92
4.2 Statement syntax 92
4.3 Number type 94
4.4 Assembler arithmetic operations 94
4.5 Program directives 95
4.6 Conditional assembly 101
4.7 Reserved word 104

Chapter 5 Debugger: a source level debugger 105

5.1 Employ breakpoints 105
5.2 To debug a project 106

Chapter 6 C Fundamental Elements 107

6.1 Comments 107
6.2 Reserved Words 107
6.3 Preprocessor Directives 108
6.3.1 #include 108
6.3.2 #define 109
6.3.3 #if, #else, #elif, #endif 110
6.3.4 #ifdef, #ifndef 110
6.4 Literal Constants 111
6.4.1 Numeric Constant 111
6.4.2 Character Constant 111
6.4.3 String Constant 111
6.5 Data Type 112
6.6 Enumeration 113
6.7 Structure and Union 113
6.8 Array 114
6.9 Pointer 115
6.10 Operators 115
6.10.1 Types of Supported Operators 115
6.10.2 Prefix of Operators 117
6.11 If-else Statement 117
6.12 Switch Statement 117
6.13 While Statement 118
6.14 Do-while Statement 118
6.15 For Statement 119
6.16 Break and Continue Statements 119
6.17 Goto Statement 120
6.18 Function 120
6.18.1 Function Prototype 120
6.18.2 Function Definition 121

Chapter 7 C Control Hardware Related Programming 122

7.1 Register Page (rpage) 122
7.2 I/O Control Page (iopage) 124

7.3 Ram Bank....125
7.4 Bit Data Type....126
7.5 Data/LCD RAM Indirect Addressing....127
7.6 Allocating C Function to Program ROM....128
7.7 Putting Data in ROM....129
7.8 Inline Assembler....130
7.8.1 Reserved Word....130
7.8.2 Use of C Variable in the Inline Assembly....131
7.9 Using Macro....132
7.10 Interrupt Routine in TCC1....132
7.10.1 Interrupt Save Procedure....133
7.10.2 Interrupt Service Routine....133
7.10.3 Reserved Common Registers Operation....133
7.11 Interrupt Routine in TCC2....139
7.11.1 Interrupt Save Procedure....140
7.11.2 Interrupt Service Routine....140
7.11.3 Reserved Common Registers Operation....140

Chapter 8 To grip Tiny C Compiler quickly 145

8.1 Create a new project....145
8.2 Add a New C file to the project....148
8.3 Add a new second file or head file to the project....148
8.4 The main() and interrupt save and service frame that WicePlus supplies....150
8.5 Developing your project, especially about interrupt....151

Chapter 9 C Compiler debug tips.... 155

9.1 Speed Up Debug....155
9.2 View corresponding Assembly code in C environment....156
9.3 Watch Defined variable in Register Window....157
9.4 Reduce cods size in some cases....158

Appendix A Assembly Error Message 159

A.1 Class M: main program errors....159
A.2 Class A: assembler errors....160
A.3 Class L: linker error....164

A.4 Class D: Debugger error 164

Appendix B C Conversion Table 167

B-1 Conversion between C and Assembly Codes..... 167

Appendix C Frequently Asked Questions about Assembly (FAQ) .. 175

C.1 Q：ICE can not connect to PC? 175

C.2 Q：The connection from ICE to PC is successful, but program dumping fails.... 175

C.3 Q：If ICE memory is checked, and the result of ICE memory is error..... 175

C.4 Q：If users try to reconnect the same ICE from PC, why does the execution become slower and slower ? 176

C.5 Q：Why can't "Step Into" be executed? 176

C.6 Q：If source files are located in Novell file server, why is the program reassembled again at each execution time? 176

Appendix D Frequently Asked Questions About Tiny C Compiler (FAQ) 177

D.1 Q: What is the maximum number of the function parameters? 177

D.2 Q: In a function, what is the maximum depth of the function call?..... 177

D.3 Q: What is the maximum array dimension as well as maximum array element? .. 177

D.4 Q: Is there any error message when the code exceeds the ROM size?..... 177

D.5 Q: In a high level interrupt subroutine, can user allocate the address in the ROM? (e.g., using "page" data type, putting "_asm{ org xxx}" before a subroutine, etc.) 178

D.6 Q: Is "static" used in the same way as in ANSI C? 178

D.7 Q: Is there any error message in case user defines too many variables in the "const" that exceeds the ROM space? 178

D.8 Q: How do I declare the variable in *.h file and using not only in one .c file? 178

D.9 Q: Should I change any program page or bank? 178

D.10 Q: May I know how many stacks I have called? 179

D.11 Q: Does C compiler just occupy 0x10~0x1F general purpose ram? 179

D.12 Q: Does C compiler support this item, such as EM78P510N? 179

D.13 Q: How do I use macro with variable? 179

D.14 Q: Does C Compiler support all assembly instruction? 179

D.15 Q: How can clear all ram in all banks? 180

User's Manual Revision History		
Doc. Version	Revision Description	Date
1.0	User's Guide initial version	2007/01/03
1.1	1. Add: A new Dialog "View setting Dialog", to see section 2.3.7c. 2. Add: Chinese Code Converter, to see section 2.7. 3. Add: How to define label, to see section 4.2.	2007/02/01
1.2	1. Add: USB ICE points for attention, to see section 1.3, 2.3.7 b, 3.2. 2. Add: Find dialog of output window, to see section 2.2.8 3. Add: New Debug Menu "Address Break Point", to see 2.3.5 4. Add: New item of "View setting Dialog" , to see section 2.3.7c	2007/04/06
1.3	1.Add: "Step Out " to see 2.3.5, 2.4, 3.8 2.Add: sub window of Program ROM at Output Window, to see 2.2.8	2007/05/08
1.4	New Menu	2007/05/31
2.0	Combine C and Assembly user manual	2007/06/05
2.1	Add To Grip Tiny C compiler quickly	2007/06/08
2.2	1. Add "Label / Function View", to see section 2.2.1. 2. Add "Customize", to see section 2.3.7e.	2007/07/03
2.3	1.Users can declare variable in common registers 0x10~0x1F. Please reference sec 7.1. 2.Users can declare bit data type in local field. Please reference sec 7.4.	2007/07/12
2.4	1. Add: Call stack View, to see section 2.2.4. 2. Add: Formation Selection. Only support C Mode, to see section 2.3.2. 3. Add: Debug Option Setting, to see section 2.3.7.	2007/07/25
2.5	1. Modify: Correct about interrupt C system back up description in Example 1, page 124	2007/09/27
2.6	1.Fix: C source code can't be written over column 512. 2. Add: Call Stack window supplement, sec. 2.2.4, page 26 3. Add: Long Delay Time description, sec. 3.2.1, page 77 4. Add: Step Over Macro description, sec. 2.3.7, page 62	2007/11/06
2.7	1.TCC2 distribution note; 2.TCC2 interrupt service routine , sec. 7.11, page 132 3. TCC2 menu, sec. 2.3.7 ,page 60 4. Add "Format select ",Edit menu, in C environment. See page 38	2008/07/10
2.8	1. Add Q & A in Appendix D; 2. Replace some pictures in sec. 2.2.6 3. Add filter in connect dialog, see sec. 3.2.1, page 76; 4. Use new code option dialog, see sec. 3.2.2, page 78	2008/10/13
2.9	1. Add: "LCD RAM" Window, to see section 2.2.8. 2. Fix: The content of Q.12, Appendix D 3. Fix the lost of reti inline assembly in example 3, EM78567, sec. 7.11.3	2008/11/20

Chapter 1
Introduction

1.1 Overview

WicePlus is ICE integrated development environment (IDE) software on Windows 2000 or Windows XP for the 8-bit microcontrollers, EM78series, of Elan Microelectronics Corporation (EMC). The WicePlus has four main modules which are editor, project, assembler and debugger. The aims of the WicePlus are to provide a friendly operation environment, powerful functions, a high-speed transmission and a stable system.

The WicePlus is project oriented IDE software. The submenus of “Editor”, “Project manager”, “Assembler” and “Debugger” are included. The main items of the Editor are “find”, “replace”, “undo”, “redo”, “cut”, “copy” and “paste.” The project manager has the function of inserting files to a project, deleting files from a project and compiling a project. The function of assembler includes “include”, “macro”, “assembly arithmetic”, “conditional assembly”, “list files” and “map file.”

Moreover, the WicePlus also offers users a lot of friendly functions including block comment, auto-updated registers, and real time line disassembler.

NOTE

■ *Please note that WicePlus can only be installed in the predefined directory (C:\EMC\WicePlus). This restriction is to prevent user from assigning an installation path that contains space char which may cause serious error while compilation.*

■ ***The file path (*.apj ,*.cpj, .dt ,.c , *.h and *.inc) CAN’T contain space in it. I f there are spaces in the path , error will occurred while compilation .***

1.2 System Requirements

1.2.1 Host Computer

The EM78 Series C Compiler requires a host that meets the following specifications**:**

- IBM PC (Pentium 100 or higher is recommended) or compatible computers
- Win2000, WinME, NT, or WinXP and Vista-32
- At least 40 MB **(**or more) free hard disk space

- At least 256MB of RAM. 512MB or more is recommended
- Mouse and USB connectors are highly recommended

1.3 Software Installation

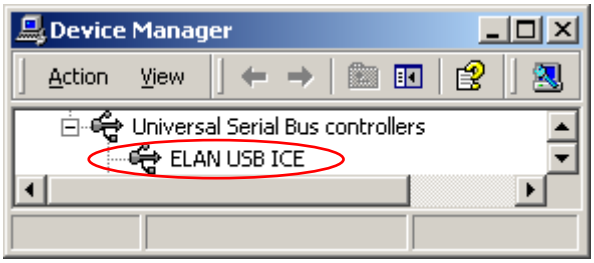
The compiler is included in WicePlus, the EM78 Series Integrated Development Environment (IDE). When installing WicePlus, the compiler will also be installed.

The drivers of ICE are included in WicePlus, too. If it is the first time to install WicePlus, because of the printer port driver (DLPortIO), you have to reboot your computer after installing WicePlus. If you use USB ICE under Windows 9X, you also need to reboot.

If it is not the first time to install WicePlus, the setup program will uninstall the previous installation then install the new one.

During installing, users cannot change the default install path.

When the operating system is searching the USB ICE hardware, please make sure the power of ICE is set to ON. You can see the EMC USB ICE by Device Manager of OS if the driver is installed and the ICE is connected correctly.



1.4 ANSI Compatibility

Compliance with the ANSI standard is limited to free-standing C to accommodate the unique design characteristics of the EM78 Series microcontrollers.

Chapter 2
WicePlus Interface

2.1 Overview

WicePlus is a project oriented integrated development environment (IDE) system that is used to edit user application programs and generates emulation/layout files for ELAN's EM78 series (8-bit) microcontrollers.

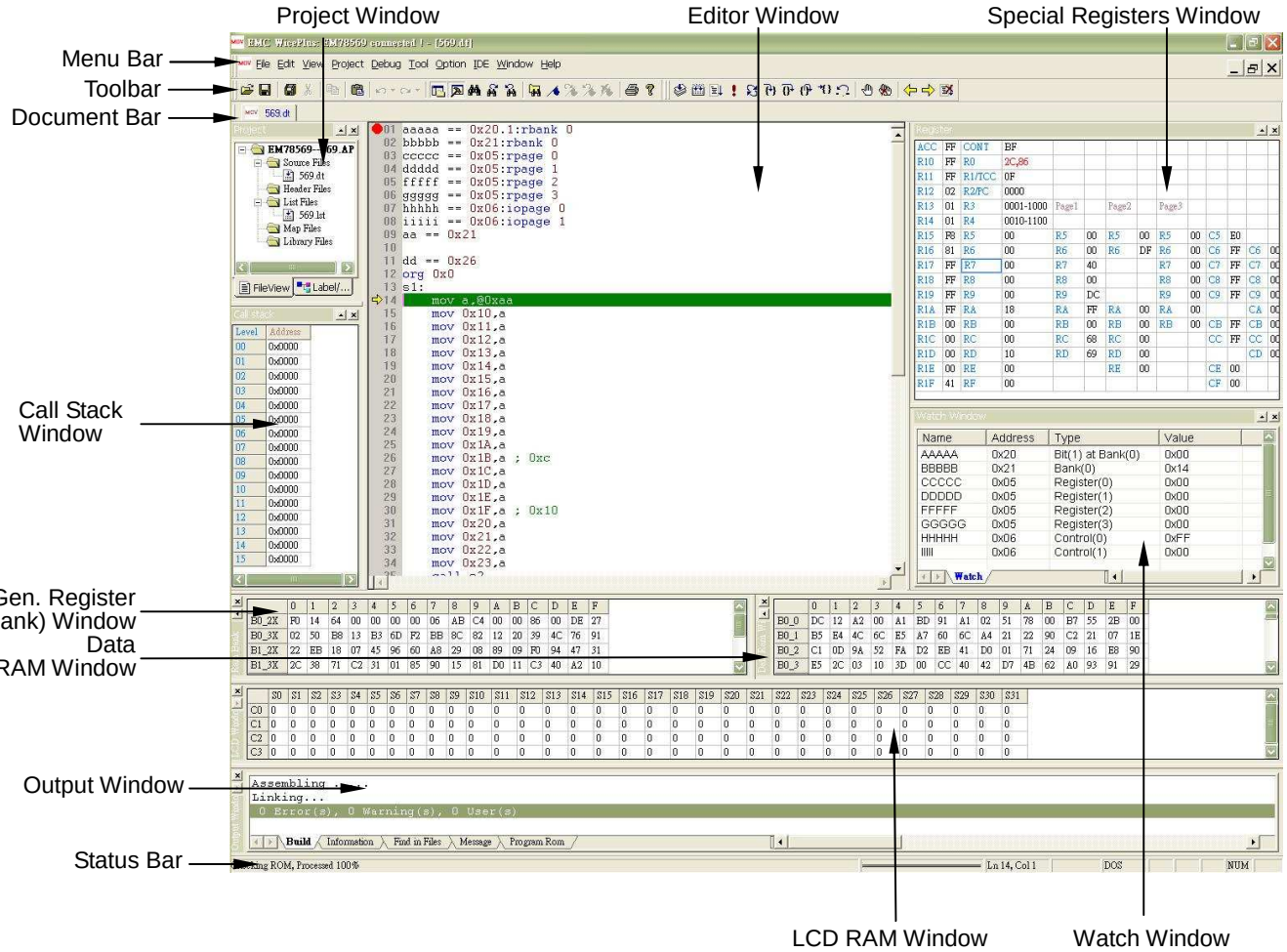


Fig. 2-1 WicePlus Main Window Layout

2.2 WicePlus Sub-Windows

The sub-window may be displayed or hidden by clicking on the pertinent window commands from the View menu (see Section 4.3.3.3)

2.2.1 The "Project" Window

There are two views in the Project Window.

1. FileView:

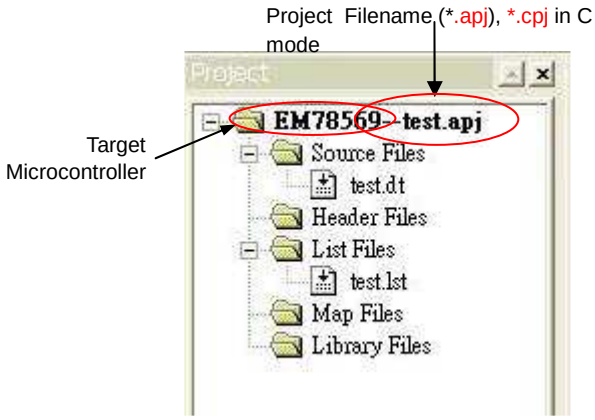


Fig. 2-2 ***Project*** *Window FileView*

The **Project** window holds the *Source*, *Header*, *List*, and *Map* files.

Where**:**

Source Files (**.dt*) – are the assembly source files that are added into the current project. In C mode, source file will be *.c file.

Header Files *(*.h)*– are the reference files required by source program.

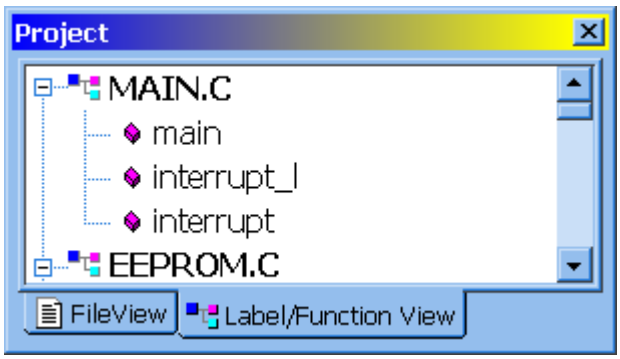
List Files *(*.lst)* – are the list files .

Map File(*.map) – are the map file .

Library File(*.bbj) – are the reference files required by source program.

The Title Bar of the **Project** window shows your current microcontroller and project filename.

2. Label / Function View



After performing a code dump, WicePlus will automatically determines the functions from C code (see figure above) and labels from ASM code. The result is displayed in the **Label/Function View** mode of the **Project** window categorized by files.

To access the **Label/Function View** window, click the **Label/Function View** tab at the bottom of the **Project** window

To find the location of the displayed label or function in the file, double-click on a label or function. WicePlus automatically pinpoints to the pertinent location in the **Editor** window (see Figure 2-2-1). At the same time, WicePlus displays the search results in the **Output** window as illustrated in Figure 2-2-2.

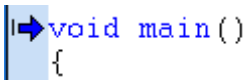


*Figure 2-2-1. Searched Label/Function Location Pinpointed in the **Editor** Window*

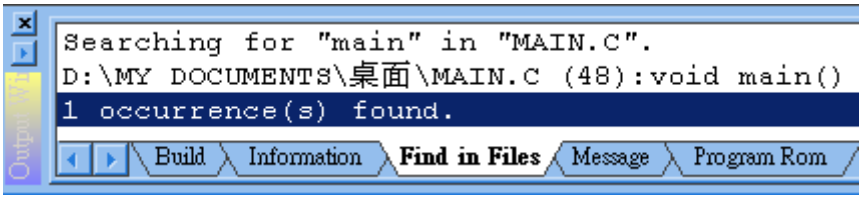
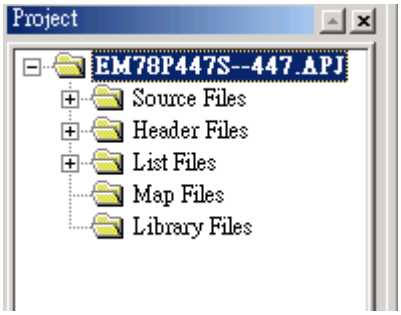


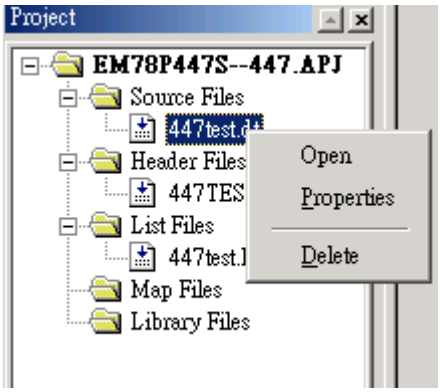
*Figure 2-2-2. Search Results Simultaneously Displayed in the **Output** Window*

If after editing the code WicePlus is unable to locate the label/function location in the **Editor** window, the **Output** window will still display a word find result for your reference.

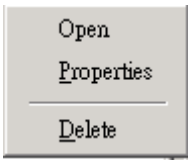
Supplement



After you have opened or created a project, click the button of the selected folder to expand and browse its contents.



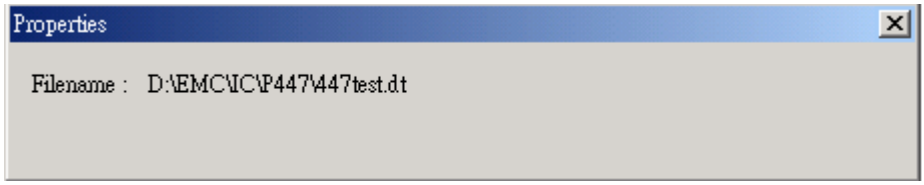
Right-click on a file in the **Project** window to display file shortcut menu.



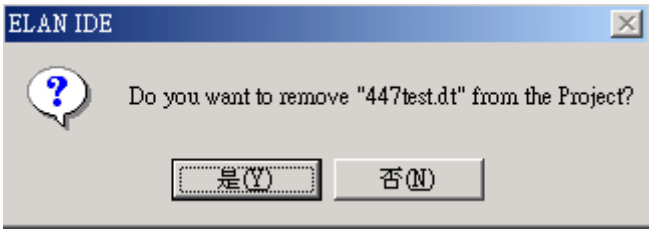
The following describes the functions of the 3 commands in the shortcut menu:

Open – Opens the selected file. For example, right click on 447test.dt, the file then opens. If the file is already opened, no action is performed.

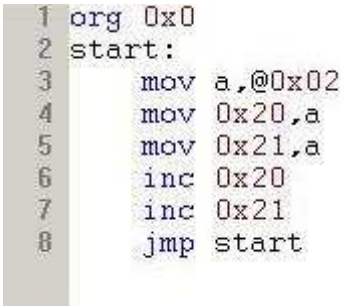
Properties – Displays the complete path of the selected file. The following figure shows the where the *447test.dt* file is located.



Delete – Removes the selected file from the current Project Source Files folder. If the file is still opened, the program closes the file before removing it. For example, if you select to remove the *447test.dt* file, a confirmation dialog will display. Click the **Yes** to remove the file from the project; otherwise click the **No**.



2.2.2 The “Editor” Window



The **Editor** window is a multi- windowed editing tool for creating, viewing, and debugging source files.

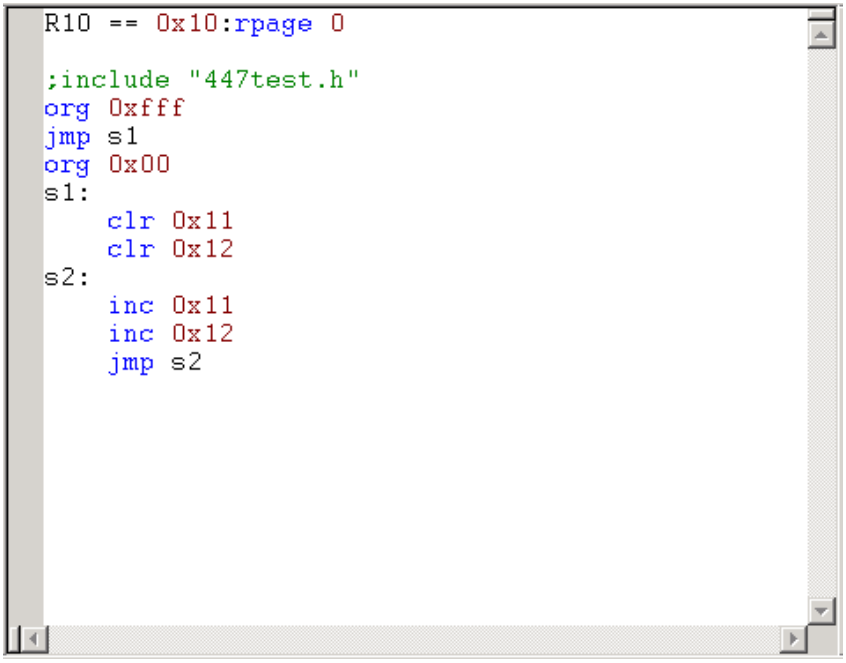
*Fig. 2-3 **Editor** Window*

The **Editor**’s major features are –

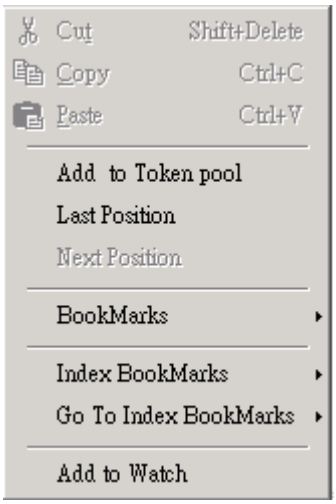
- Unlimited file size
- Multiple files can be opened and displayed at the same time
- Insert (overstrike) mode for editing
- Undo/Redo
- Clipboard support (text can be cut, copied, moved, and pasted onto the clipboard using a keystroke)
- Drag and drop text manipulation (highlighted text can be dragged and dropped between any of the IDE windows)

Supplement

The figure below shows a typical **Editor** window displaying contents of an opened source file (*447test.dt*). Assembly keywords are shown in blue, comments & comments symbol are in green, values are in brown, and the rest are shown in black.



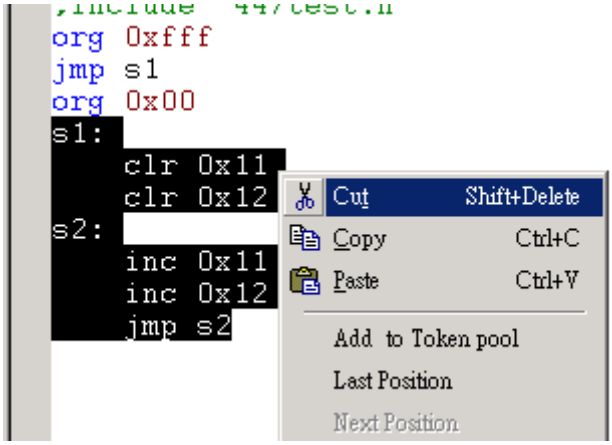
Right-click anywhere within the **Editor** window to display the **Edit** shortcut menu.



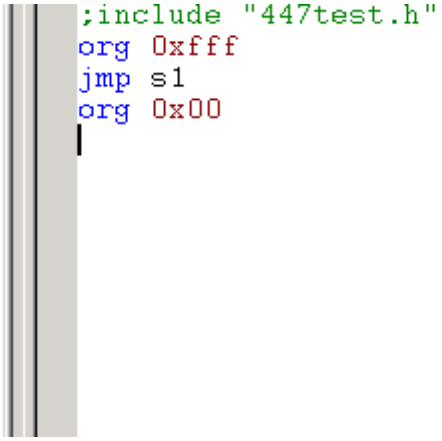
The following describes each command in the shortcut menu:

Cut – Remove the selected text from current location and move (paste) it into another location.

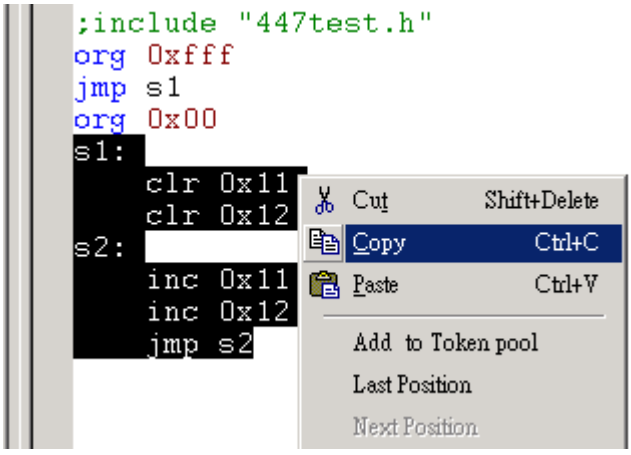
First, select the desired text range you want to move and then right-click within the selected text. With the shortcut menu on display, click **Cut** command from the menu. The selected text are then removed from the **Editor** window (as shown in the figure below) and temporarily stored in the clipboard. Proceed to paste the text into your target location.



The figure below shows the result after the cut operation.

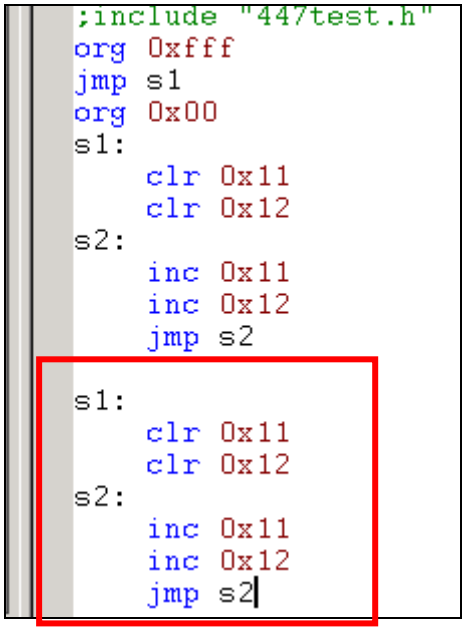


Copy – First, select the desired text range you want to copy and then right-click within the selected text. With the shortcut menu on display, click **Copy** command from the menu. A copy of the selected text is then temporarily stored in the clipboard. Proceed to paste the text into your target location.



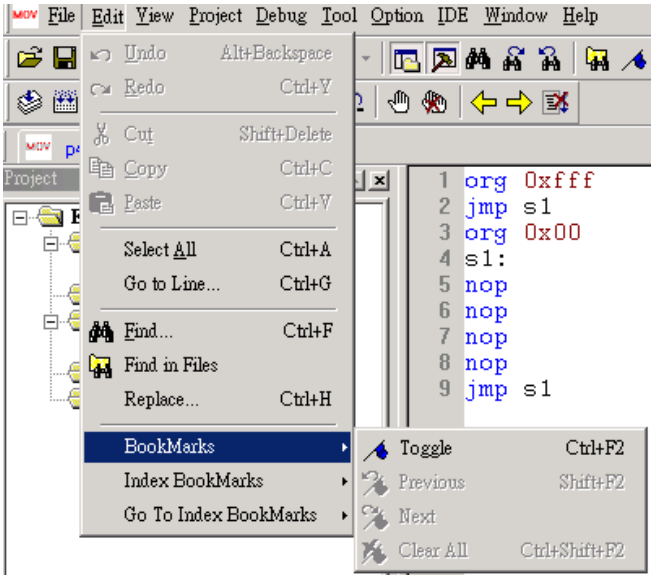
Paste – Insert the selected text (that has been recently cut or copied into clipboard) into the target location. The following example shows the paste result of the text at the bottom (shaded)

of the same page in the Editor window.



Bookmarks – Insert markers to specific lines that you may wish to return to at later time.

Step 1 To insert bookmark, place cursor on the selected line, then click **Edit → BookMarks → Toggle** from the Menu bar (or directly press the shortcut keys CTRL + F2).

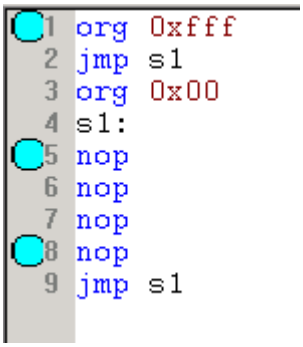


Step 2 Go to bookmarked lines. For example, if you have previously bookmarked Lines 1, 5, and 8 (as shown in the sample figure below) and want to return to the lines, access **Toggle** command again. The **Previous**, **Next**, and **Clear All** commands become active this time.

Click **Previous** to go upward to the bookmarked line previous to current position (or directly press the shortcut keys SHIFT + F2).

Click **Next** to go downward to the bookmarked line next to current position (or directly press the shortcut keys CTRL + SHIFT + F2).

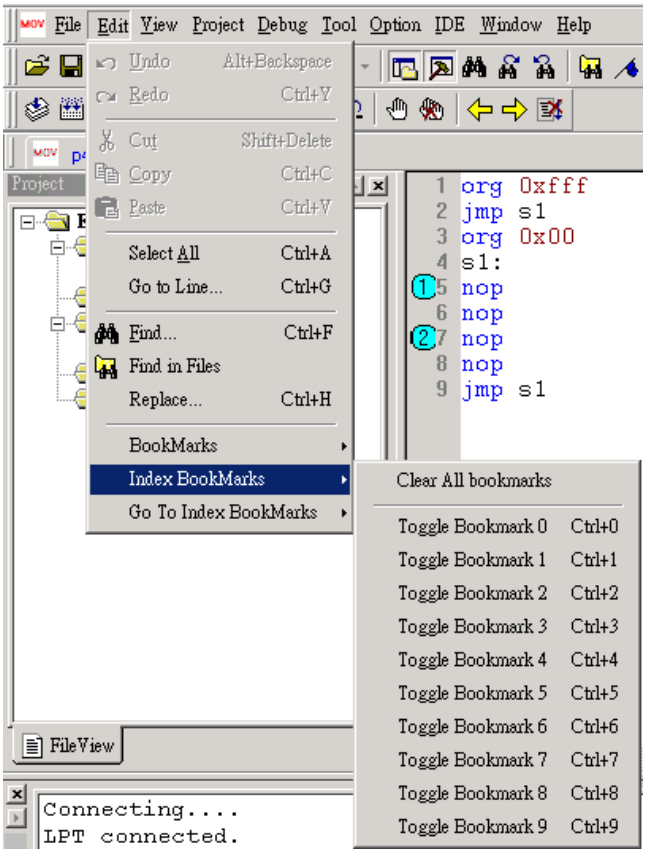
Click **Clear All** to remove all existing bookmarks.



Index BookMarks – Insert bookmarks with index numbers. With indexed bookmarks, you can directly access to the bookmarked line you wish to return to.

To insert bookmark, place cursor on the selected command line, then click **Edit → Index BookMarks → Toggle BookMark *x*** (where "***x***" is the 0 ~ 9).

You can also directly press the shortcut keys CTRL + *x* (where "*x*" = to be assigned index bookmark number 0 ~ 9).



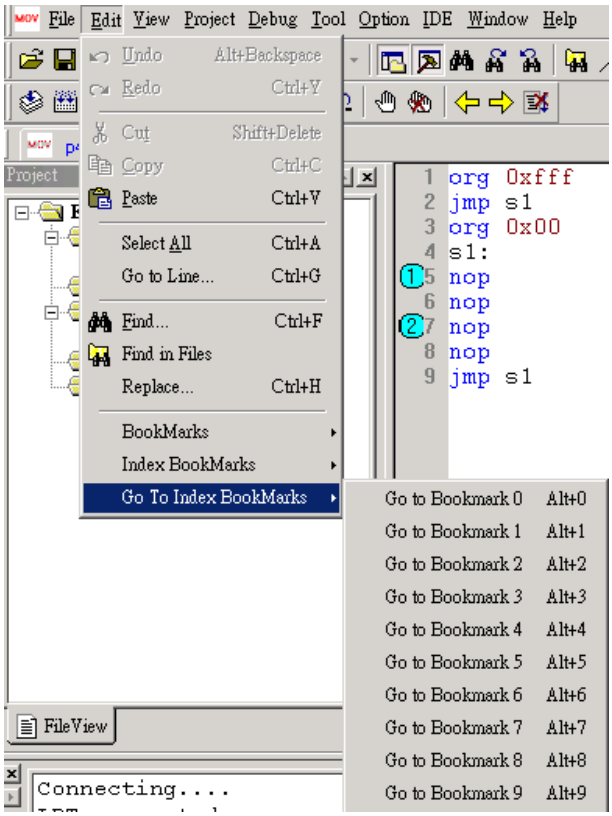
The following figure shows an example when if you set index bookmark 1 at line 5 and index bookmark 2 at line 7.

```
1  org 0xfff
2  jmp s1
3  org 0x00
4  s1:
5  nop
6  nop
7  nop
8  nop
9  jmp s1
```

Go To Index BookMarks - Go to a particular index bookmarked line.

From Menu bar, click **Edit → Go To Index BookMark**. Then from the resulting submenu, click on the index bookmark number you want access.

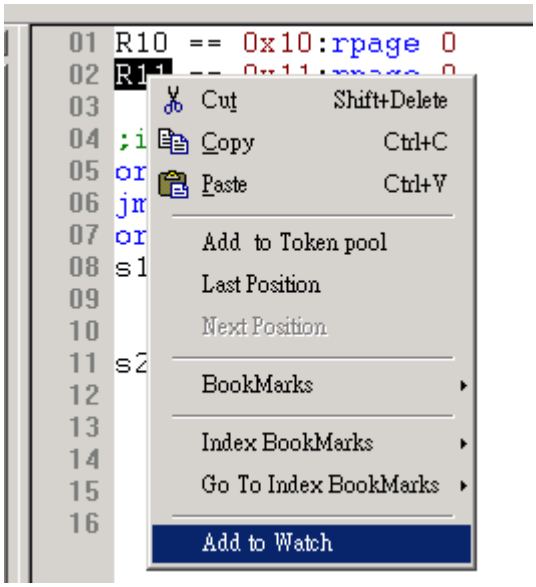
You can also directly press the shortcut keys ALT + x (where "x" = the target index bookmark number 0 ~ 9).



Add to Watch –Watch the data change at the register locations in the **Watch** window during debugging.

After dumping project to ICE, right-click on a temporary register name (followed by "==" symbol and **register page** or **ram bank** or **control register page**) as indicated in the example figure below. From the resulting pop-up menu, click **Add to Watch**. Then observe Register **R11** being added into the **Watch** window. Real-time changes of the data during debugging can

be observed from this window.



When right-clicking on a temporary register name with double–equal characters "==", but without **register page/ram bank/control register page**, the **Watch Dialog** will display instead. See Section 2.2.5, *Watch Window* for further details.

2.2.3 The "Special Register" Window

When value changes, it is shown in red

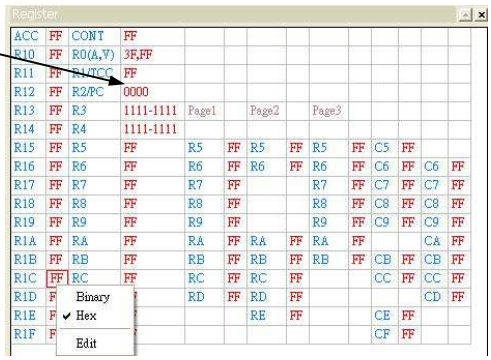

Register
ACC FF CONT FF
R10 FF R0(A,Y) 3F,FF
R11 FF R1/PCC FF
R12 FF R2/PC 0000
R13 FF R3 1111-1111 Page1 Page2 Page3
R14 FF R4 1111-1111
R15 FF R5 FF R5 FF R5 FF R5 FF C5 FF
R16 FF R6 FF R6 FF R6 FF R6 FF C6 FF C6 FF
R17 FF R7 FF R7 FF R7 FF C7 FF C7 FF
R18 FF R8 FF R8 FF R8 FF C8 FF C8 FF
R19 FF R9 FF R9 FF R9 FF C9 FF C9 FF
R1A FF RA FF RA FF RA FF RA FF CA FF
R1B FF RB FF RB FF RB FF RB FF CB FF CB FF
R1C FF RC FF RC FF RC FF CC FF CC FF
R1D FF RD FF RD FF CD FF
R1E FF RE FF CE FF
R1F FF CF FF
Binary
Hex
Edit

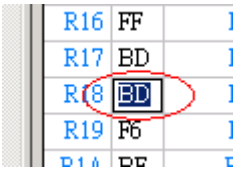
Fig. 2-4 ***Special Register*** *Window*

The **Special Register** window shows the updated contents of the registers and I/O control register depending on the MCU type.

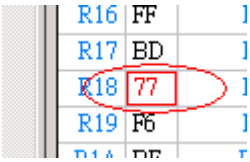
When closes the Windows, will not read to the hardware any concerns this Windows's contents. Except some special registers.

Supplement

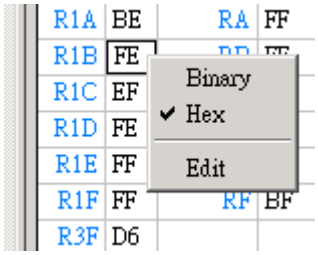
To change a special register value, double-click on the selected value (**BD** in the example figure below).



With the existing value highlighted, key-in the new value (**77** in the example figure below). Observe the new value changes to red when you click anywhere within the **Ram Bank** window.

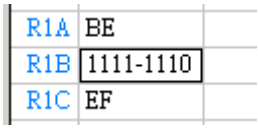


Right-clicking on the selected value will also generate a pop-up menu containing commands for editing the selected register value.

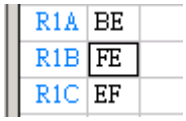


The following describes each of the menu commands**:**

Binary – Changes the register value from hex to binary format. If the value is already in binary, this command is prefixed with a check mark (✓).

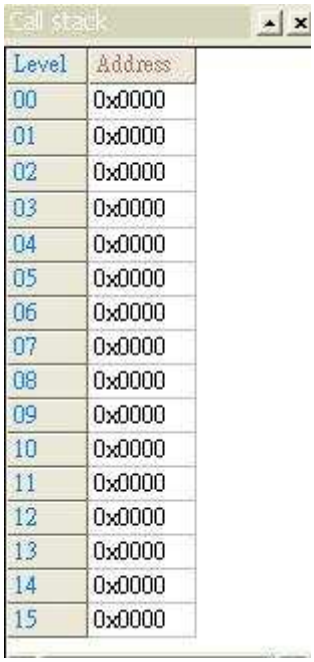


Hex – Change the register value from binary to hex format. If the value is already in hex, this command is prefixed with a check mark (✓).



Edit – Revise the existing register value. This is the same as double-clicking on the value as explained above. However, using this **Edit** function is preferable.

2.2.4 The “CallStack” Window



The **CallStack** window shows the updated contents of call stack.

This Windows's content, responds the hardware actual content really.

When closes the Windows, will not read to the hardware any concerns this Windows's contents.

Supplement

In general, a stack does not have an initial value. When you press F6, WicePlus resets all stack cells to 0x0000. Due to ICE hardware design constraint, if a stack is full (see Figure 2-5-1) during program execution and returns to the calling sub-routine, reading from the stack again will display the result as shown in Figure 2-5-2. This may also affect higher-level sub-routines (as shown in Figure 2-5-3).

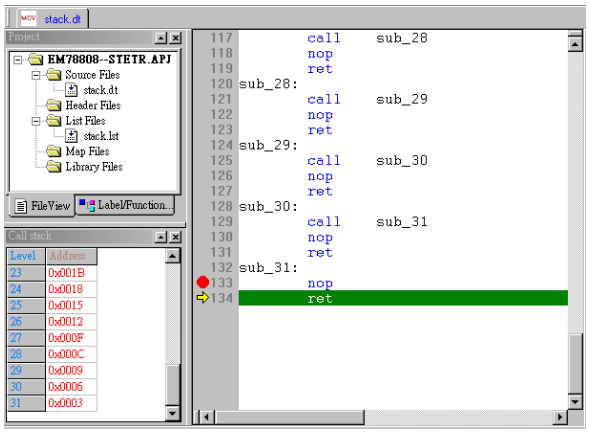


Fig. 2-5-1 All stack levels are full (as indicated by the breakpoint setting)

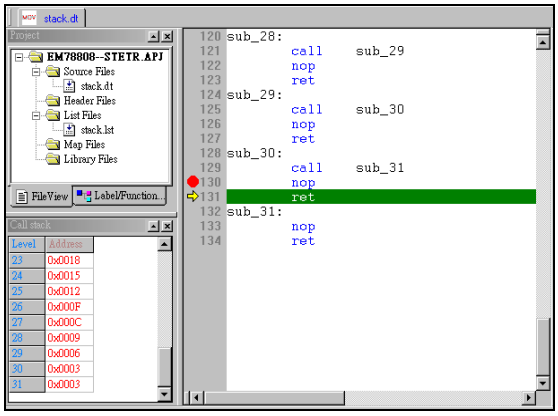


Fig. 2-5-2 Return to the preceding stack level but note that the last (next) stack level still shows 0x0003 (the value of the last stack level when all stack were full)

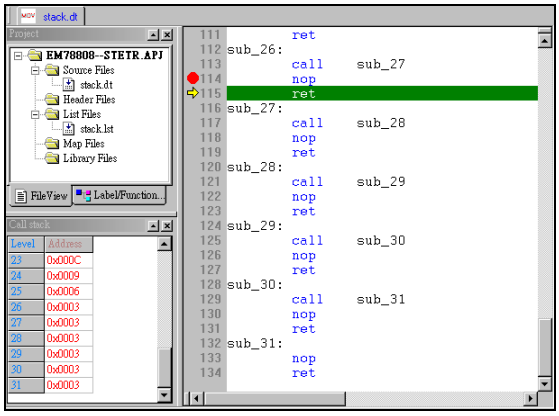


Fig. 2-5-3 Return to an earlier level (several levels) of stack. However, note that all the subsequent stack levels still display 0x0003

When you perform a step-by-step execution (F7), WicePlus will compare the last stack value with the stack value after the “ret” action to identify the current level position of the stack. It then changes the value of the last stack level value from 0x0003 to 0x0000 (see Figure 2-5-4).

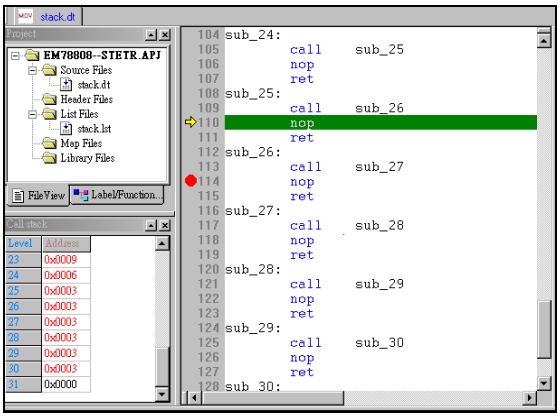


Fig. 2-5-4 Using Step-by-Step (F7) execution in WicePlus to change the last stack level value

To set the values of the higher-level of stacks to 0x0000, continue to perform Step-by-Step

execution (see Figure 2-5-5). This will not affect normal stack operation.

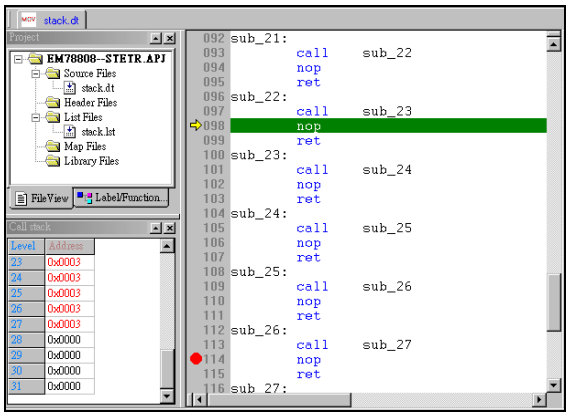


Fig. 2-5-5 Using Step-by-Step execution to change the values of the higher-level stacks

Currently, WicePlus is unable to determine the correct stack level position at which the ICE stops. This is due to the ICE hardware limitation which unable the WicePlus Call Stack window to display the correct stack level position properly. However, this does not affect the actual normal stack operation.

Also note that after completing the Go or FreeRun command execution, if all stack level values in Call Stack window display the same value, this could be caused by either the program has entered the first level of the call function or the program has not entered any function t all. Under this condition, it is not recommended to use the StepOut command. Otherwise, the program will jump to the first level address in the Call Stack window.

2.2.5 The “General Registers (Ram Bank)” Window

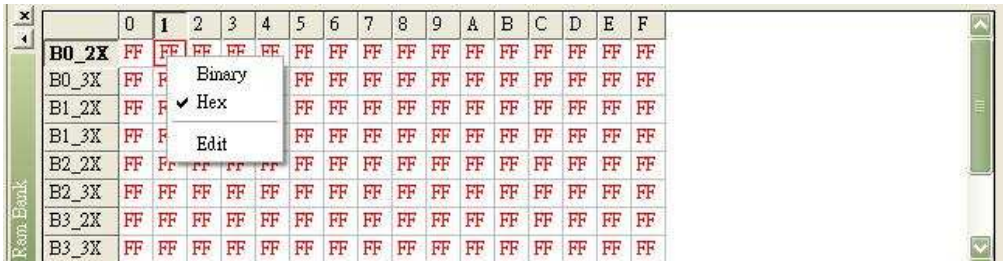
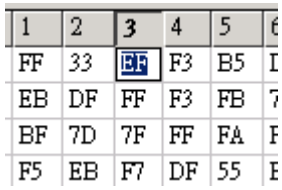


*Fig. 2-6 **General Registers (Ram Bank)** Window*

The **General Registers (Ram Bank)** window shows the updated contents of the common ram bank registers. When closes the Windows, will not read to the hardware any concerns this Windows's contents.

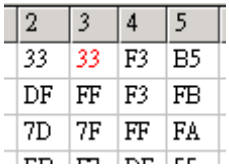
Supplement

To change a general register value, double-click on the selected value (**EF** in the example figure below).



1	2	3	4	5	6
FF	33	EF	F3	B5	D
EB	DF	FF	F3	FB	7
BF	7D	7F	FF	FA	F
F5	EB	F7	DF	55	E

With the existing value highlighted, key-in the new value (**33** in the example figure below). Observe the new value changes to red when you click anywhere within the WicePlus window.



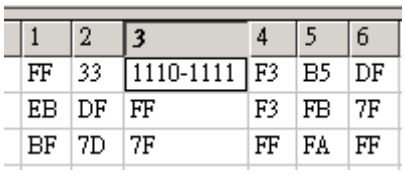
2	3	4	5
33	33	F3	B5
DF	FF	F3	FB
7D	7F	FF	FA

Right-clicking on the selected value will also generate a pop-up menu containing commands for editing the selected register value.

	0	1	2	3	4	5	6	7	8
B0_2X	5F	FF	33	EF				FB	FD
B0_3X	F3	EB	DF	FF				DF	9D
B1_2X	1D	BF	7D	7F				FF	DF
B1_3X	FF	F5	EB	F7				7F	2F
B2_2X	F7	FF	76	FD	77	DC	EF	F7	FF

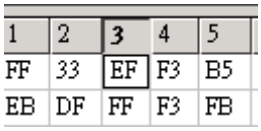
The following describes each of the pop-up menu commands**:**

Binary – Changes the register value from hex to binary format. If the value is already in binary, this command is prefixed with a check mark (✓).



1	2	3	4	5	6
FF	33	1110-1111	F3	B5	DF
EB	DF	FF	F3	FB	7F
BF	7D	7F	FF	FA	FF

Hex – Change the register value from binary to hex format. If the value is already in hex, this command is prefixed with a check mark (✓).



1	2	3	4	5
FF	33	EF	F3	B5
EB	DF	FF	F3	FB

Edit – Revise the existing register value. This is the same as double-clicking on the value as explained above. However, using this **Edit** function is preferable.

2.2.6 The “Watch” Window

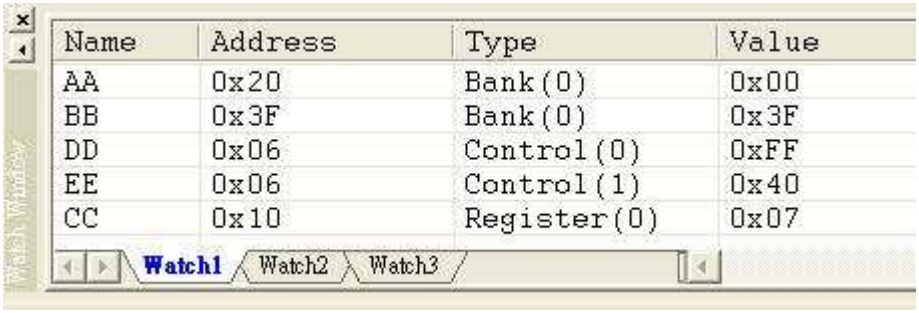


*Fig. 2-7 **Watch** Window*

In the **Watch** window, you can add variables that are declared in assembly file. The **Watch** window will show the defined variable information, such as name, contents, bank, and address. Please reference to sec. 4.2 for assembly code.

The step to add a variable to watch window:

1. Reverse the variable. (in such case is “aa”)
2. Click right button of mouse, and a menu popup.
3. Select “Add to Watch” item.

Supplement

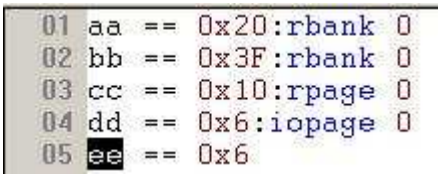
To view real-time change of the selected register values from the **Watch** window during debugging, the register values have to be entered into **Watch** window. To do so, do one of the following (three methods are available)**:**

Method 1. Directly right-click the selected register value from Editor window

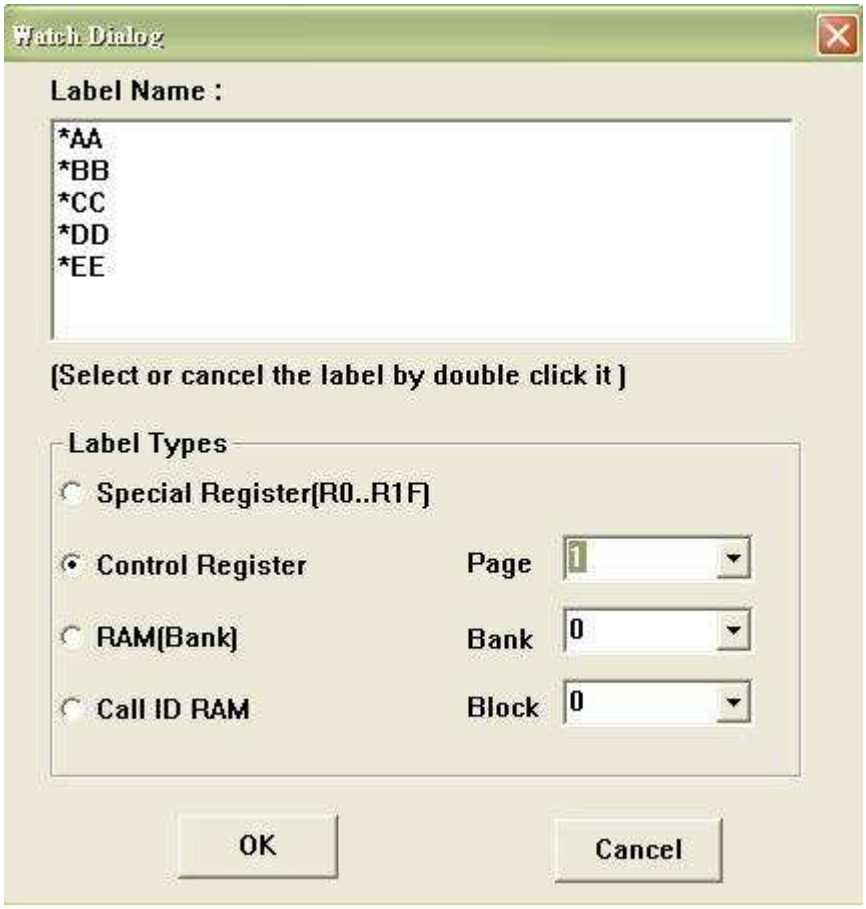
Step 1 Right-click a register value (or variable) from the **Editor** window. From the resulting pop-up menu, click **Add To Watch** command.

If the selected value contains **register page** (or **ram bank** or **control register page** (as shown in the above example figure), the register values is directly displayed in the **Watch** window (see Section 2.2.2, *Editor Window → "Add to Watch")*.

If it does not contain **register page/ram bank/control register page** (see example below), the **Watch Dialog** will display instead.

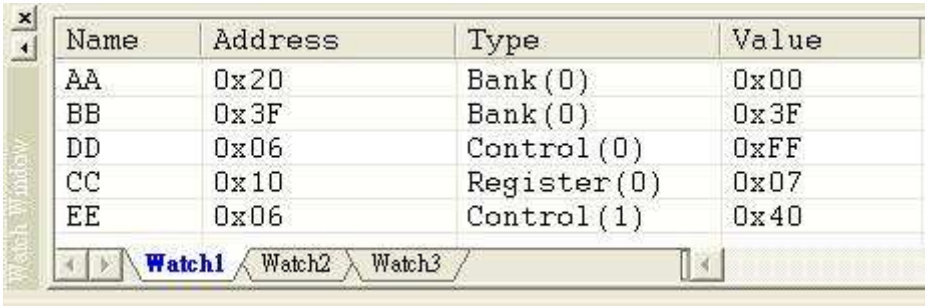


Step 2 From the **Label Name** box of **Watch Dialog** window, select and double-click a variable label name you wish to assign. Then from the **Label Types** options, select the variable type to be displayed, i.e., **Special Register**, **Control Register**, or **RAM(Bank)**.



Step 3 Click the **OK** to add the selections into the **Watch** window.

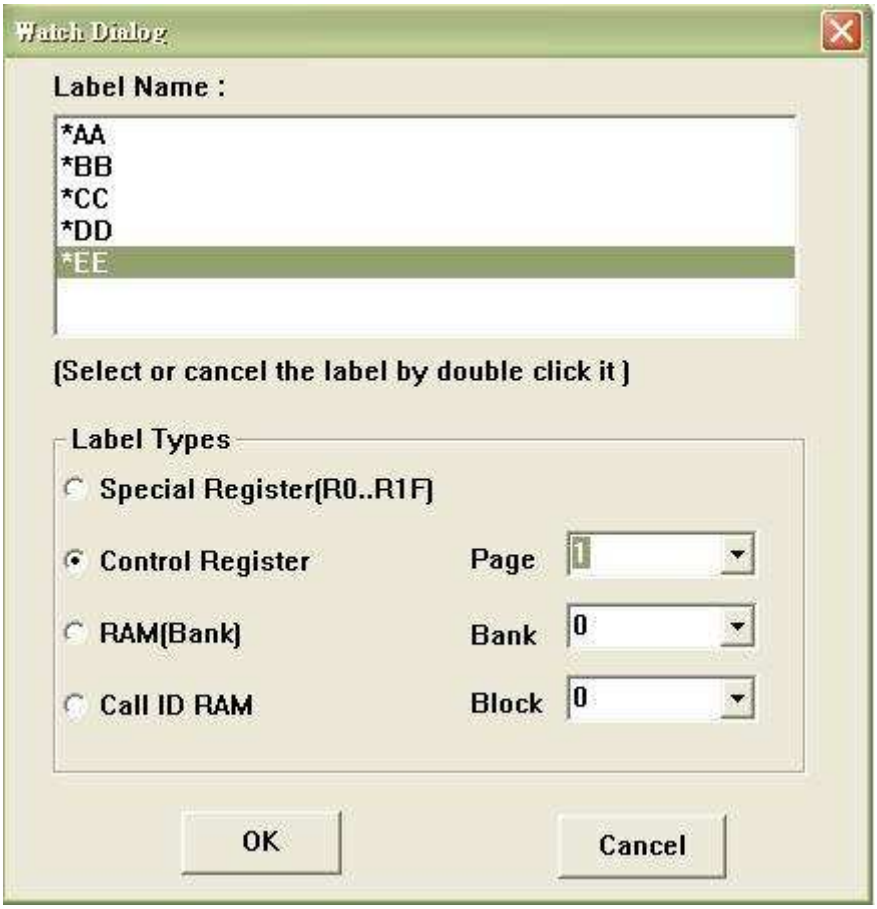
NOTE: The variables displayed in the **Label Name:** list box of the **Watch Dialog** are the same variables (without **register page/ram bank/control register page**) that you have defined in the program code using the double–equal characters "==". Double-clicking a variable name will add or clear the asterisk "*". An asterisk "*" prefixed to variable name indicates that the variable is selected. Click **OK** to add the selected variable to the **Watch** window.



Method 2. Accessing "Debug" → "Add Label to Watch" from menu bar

Step 1 Select a register from the **Editor** window and from menu bar click **Debug → Add Label to Watch**.

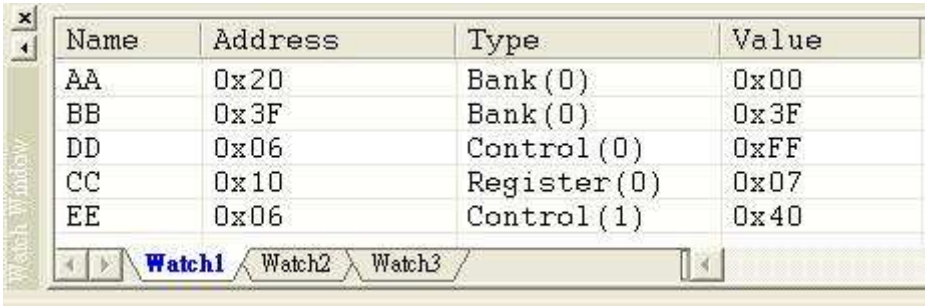
Step 2 From the resulting **Watch Dialog** window, select and double-click a label name. Then from the **Label Types** options, select the variable type to be displayed, i.e., **Special Register**, **Control Register**, or **RAM(Bank)**.



Step 3 Click the **OK** to add the selections into the **Watch** window.

NOTE: The variables displayed in the **Label Name:** list box of the **Watch Dialog** are the same

variables (without **register page/ram bank/control register page**) that you have defined in the program code using the double–equal characters "==". Double-clicking a variable name will add or clear the asterisk "*". An asterisk "*" prefixed to variable name indicates that the variable is selected. Click **OK** to add the selected variable to the **Watch** window.



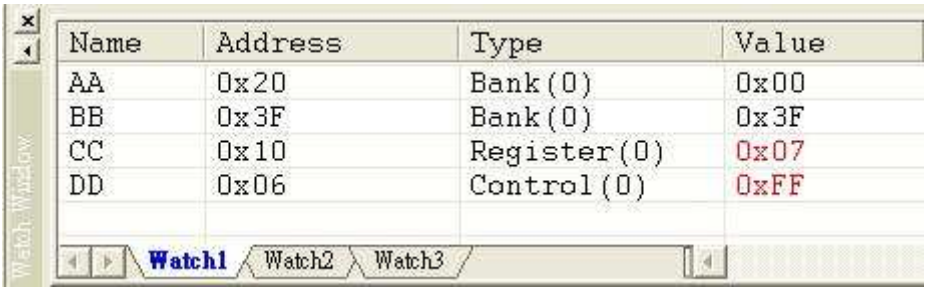
Method 3. Accessing "Option" → "View" from menu bar

Step 1 Click **Option → View Setting** and from the resulting View Setting dialog, select the **Add defined label to watch automatically** option: There are three sort options in WicePlus version 2.6 or later. Labels added into watch window will sort by these three options automatically. Now we illustrate sort by name.

Step 2 All variables with **page location data** from the **Editor** window are automatically formatted into labeled format as shown in the example below.

```
01 aa == 0x20:rbank 0
02 bb == 0x3F:rbank 0
03 cc == 0x10:rpage 0
04 dd == 0x6:iopage 0
05 ee == 0x6
```

Step 3 After code dump, the variables are automatically included in the Watch window.



NOTE: 1.To remove a variable from the **Watch** window, select the variable and press DELETE from the keyboard.

2. If checked box **Add defined label to watch automatically** was checked, labels added into watch window will sort by these three.

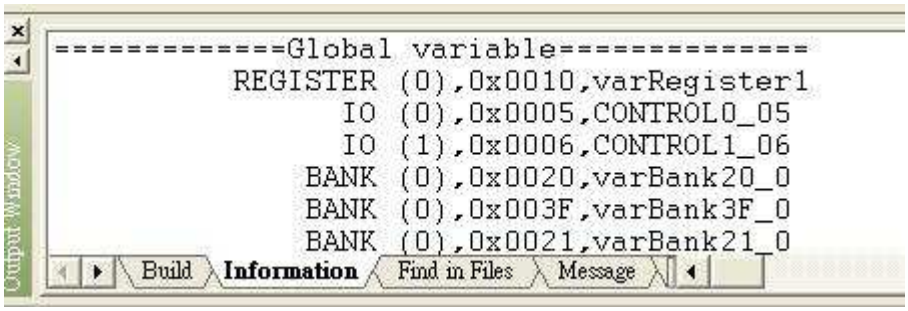
3 .The label placed in the end at **Watch** window when user manual operations in previous descriptions.

4. To execute Dump(F3) at C Mode. All global labels display at **Information** sub-window of **Output** window.

4.1. First field is the type of variable and page number.

4.2. Second filed is the register's or Rom's address of variable.

4.3. Third field is the name of variable.



2.2.7 "Data RAM" Window

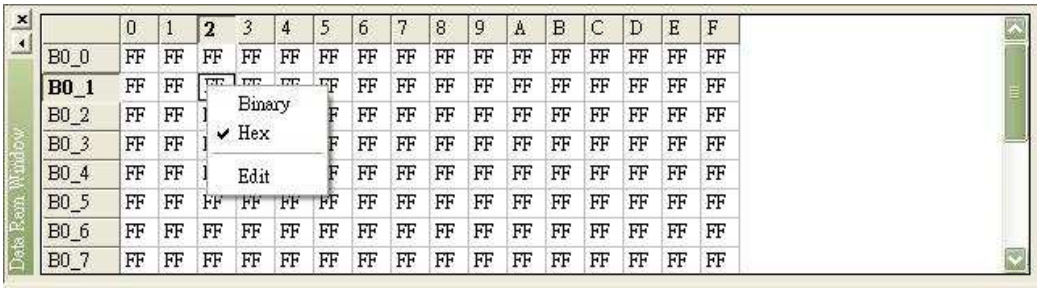

Data Ram Window
0 1 2 3 4 5 6 7 8 9 A B C D E F
B0_0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B0_1 FF FF
B0_2 FF FF
B0_3 FF FF
B0_4 FF FF
B0_5 FF FF
B0_6 FF FF
B0_7 FF FF
Binary
Hex
Edit

Fig. 2-8 ***Data RAM*** *Window*

The **Data RAM** window is accessible only if RAM is available form the target microcontroller currently in use. The **Data RAM** window shows the contents of the data RAM. When closes the Windows, will not read to the hardware any concerns this Windows's contents.

2.2.8 “LCD RAM” Window

With the IC that supports LCD installed and in used, open the **LCD RAM** window and increase the height of the **LCD RAM** window until the window looks as shown in *Figure 2-9.*

When closes the Windows, will not read to the hardware any concerns this Windows's contents.

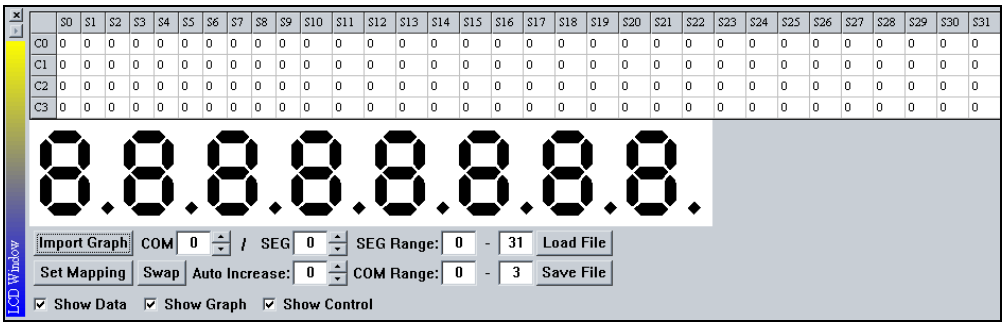


*Fig. 2-9 **LCD RAM** Window*

The **LCD RAM** window consisted of four sections as described below:

1. Data Pane:

   This pane displays the contents of the **LCD RAM**. “**Cx**” denotes the LCD signal “COM x.” “**Sx**” denotes LCD signal “Segment x.”

   To modify the contents of the LCD RAM element, double click on the selected element (grid block). The color of the elements will change from pink (1) to white (0) and vise-versa. Any related messages will be shown in the **Output** window.

2. Graphic Pane:

   This pane displays the status of the loaded BMP graphic. If no BMP graphic is loaded, no display is shown.

3. Control Pane:

   This pane controls the link between the BMP graphic and the data, as well as setting the COM/SEG values for each graphic segment.

4. Pane Selection Bar:

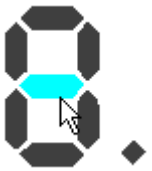
   You can select whether to display any or all of the three panes described above.

The following is the step-by-step procedure on how to load the graphic display and define their segment settings:

(1) Click **Import Graph** button to load the BMP file. The program will automatically convert the graphic file into black and white colors and determines which black pixels are linked together to establish segments.

(2) Click **Set Mapping** button and the black segments fades to gray to indicate that the system is now in Mapping Mode. At the same time, the "**Set Mapping**" button label changes to "**Done.**" Click the **Done** button only after all mapping setting processes are completed.

(3) Define COM/SEG values in the Control Pane and point at a segment to apply the value. Observe the segment turns to blue color. This indicates that the pointed segment is now active.



(4) Click on the active segment to enter the Control Panel defined COM/SEG values into the active segment. The example shows the defined COM & SEG values are "0."

(5) Move the pointer away from the segment and the segment turns into green. This indicates that the system has already saved the COM/SEG values for the particular segment.

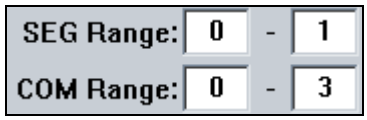
(6) Then move the pointer to other segments and repeat the steps as described above to redefine and change their respective COM/SEG values.

An automatic and faster way of re-defining and setting of COM/SEG values is explained in the next topic.

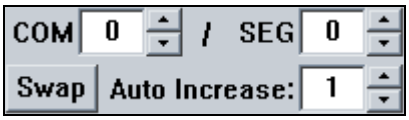
(7) After all segment values are set, click **Done** (the original **Set Mapping**) button. Then run the program to test the settings, e.g., set some breakpoints and execute "**GO"** or use the "**Continue step into.**" View the results and check for error.

The Step 6 above shows how to manually define and change the COM/SEG values for each segment. This method is okay if there are only a few segments involved. However, if a large number of segments are involved, the task becomes complicated and time consuming. The following steps explain a more efficient way of defining a large number of segment values:

(1) From the Control Pane, define the SEG Range as 0~1, and the COM Range as 0~3. Consequently, the segment values will be set according to these defined ranges.

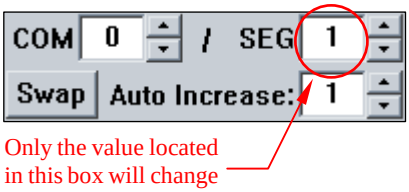


(2) Define COM/SEG values to "0" and Auto Increase value to "1".



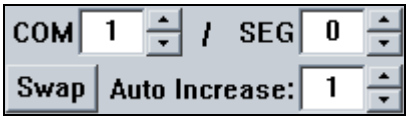
(3) Point to the segment where the COM/SEG value (0/0) is to be applied. Click and you should see the segment set as "0/0".

(4) Observe that as soon as the "0/0" value is entered on the segment, the Control Pane SEG value is automatically increased by "1". Note that only the value located at the right box will change.



(5) Thus, there is no need to manually change the Control Pane COM /SEG value when setting "0/1" value for the next segment. Just directly click on the segment.

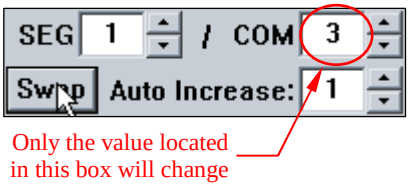
(6) Due to the fact that the SEG Range is set at 0~1 (see Item 1 above), the next or 3rd segment you click cannot increase its SEG value to "2." Therefore, the system automatically allocates the auto-increase value of "1" to COM and sets the SEG value back to "0."



(7) Therefore, all you need to do is continuously click on the remaining segments to set their COM/SEG values within the defined range.

In addition to the above, other convenient functions of the LCD RAM window are also available and are describe below together with its respective notes:

(1) If you wish to automatically apply the Auto Increase value to COM (instead of SEG), click **Swap** button and observe SEG & COM swaps positions.



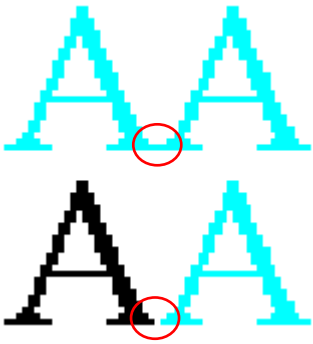
(2) After clicking **Swap**, also observe that the system has at the same time, switched the positions of all values (from COM/SEG to SEG/COM) on the set segments. Compare the figure at right to the one in Item 7 above and see the difference.

(3) After setting all the relative values to affected segments, it is recommended you click **Save File** button to save the LCD Simulator Data File (*.LCD). Otherwise, when you want to use the program the next time, you will need to define and set the values again.

(4) Since the LCD Simulator Data File already includes the imported graphic files, you do not have to clock the **Import Graphics** button the next time you want to use the saved *.LCD file. Just click **Load File** button to load the data file.

(5) The program determines which black pixels are linked together to establish a segment. However, when 2 or more separate segments constitute a non-separable object or character (e.g., *i*, *j*), all integral segments should be assigned with the same COM/SEG value. Otherwise, the program will assume that there are 2 or more separate segments.

(6) As stated above, the program determines which black pixels are linked together to establish a segment. However, when 2 or more segments which are ought to be separate and set with different COM/SEG values, are somehow joint together at one point, such graphic file should be modified to break up their link. Otherwise, the program will wrongly assumed that there is only one single segment,



2.2.9 "Output" Window

Fig. 2-10 ***Output*** *Window*

The **Output** window displays messages indicating the results (including errors) of the project compiling just performed, such as assembler, linker, trace log history, and debugging. The window consists of four tab sub-windows, namely**;** **Build**, **Information**, **Find in Files**, **Message**, and **Program Rom,** where**:**

Build – displays assembler/linker related messages and trace logs. Double click on the error message to link to the corresponding program text line where the source of error occurs. The pertinent source file is automatically opened in the **Editor** window if it is not currently active.

Information – displays debugging related ROM and RAM Bank memory usage information.

Find in Files – allows you to find identical string (selected from an active file) in other active or inactive files in your folder. Lines containing the identical string will display on the **Output** window complete with its source filename and directory.

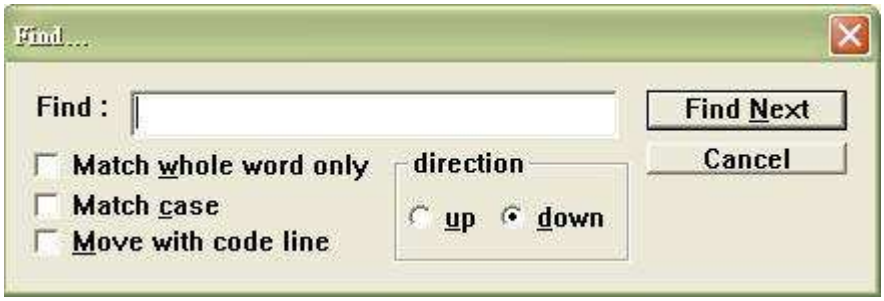
Message – displays the debugging related changes to the **LCD RAM** window.

Program Rom – displays the content of program ROM after dump.

The step to use find dialog of output windows:

1.Click right button of mouse, and a menu popup.

2.Select “Find” item.



Find: Input a word you want to find out.

Find next: Click the button to find next word which be matched.

(Hotkey: Alt + N)

Cancel: Cancel the dialog.

Up: Find forward. (Hotkey: Alt + U)

Down: Find backward. (Hotkey: Alt + D)

Match whole word only: Match whole word only. (Hotkey: Alt + W)

Match case: Match upper or lower case. (Hotkey: Alt + C)

Move with code line—when select the matched line in trace log, mark code line respond to the line in the trace log.

(Hotkey: Alt + M)

*Hot key work when dialog active.

*When WicePlus active, find forward: Ctrl + R; find backward: Ctrl + Q

2.3 WicePlus Menu Bar and its Commands

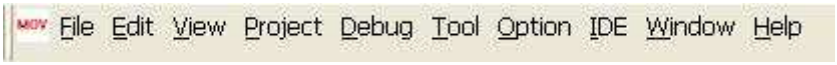


Fig. 2-11-1 ***Menu*** *Bar*

2.3.1 File Menu

Fig. 2-11-2 ***File*** *Menu*

New…	Create a new project or source file
Open…	Open an existing document or project
Close	Close the active document or project
Save	Save current active document
Save As	Save current active document with new filename
Save All	Save all opened documents
Open/Save/Close Project	Open/Save/Close the active project
Print	Print active file
Print Preview	Preview printed format of active file
Print Setup…	Define printer settings
Recent Files	View the record of the recently used file
Recent Projects	View the record of the recently used project
Exit	Exit from WicePlus Program

2.3.2 Edit Menu

*Fig. 2-12 **Edit** Menu*

Undo	Cancel the last editing action
Redo	Repeat the last editing action
Cut/Copy/Paste	Same as standard clipboard function
Select ALL	Select all contents of the active window
Go to Line…	Move cursor to the defined line number within the active window
Formation Selection	Formats the selection using smart indent settings **(Only support C)**
Find…	Find the defined strings in the active window
Find in Files	Find the defined string in the active and non-active files
Replace…	Same as standard "find and replace" editing functions
Bookmarks	Bookmark the line at cursor position
Toggle	bookmark the line at cursor position
Previous	jump to next bookmark from cursor position toward the end of file
Next	jump to next bookmark from cursor position toward the beginning of file
Clear All	clear all bookmarks
Index Bookmarks	Clear all bookmarks or assign an index value (0~9) to the bookmarks in order to easily access (jump) them using the "**Go to Index Bookmarks**" command below
Go to Index Bookmarks	Jump to bookmark with "x" index value

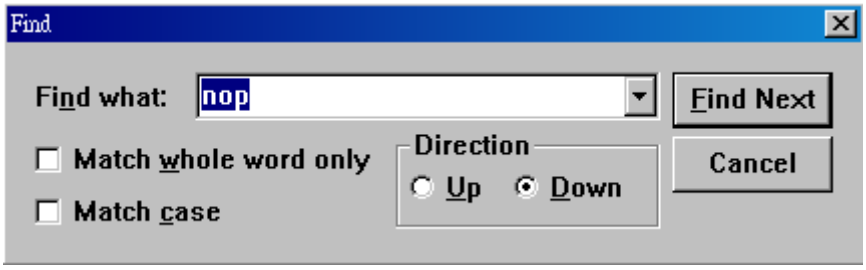
Supplement

Find function searches for a defined string in the Editor window.

Method 1. Accessing "Edit" → "Find…" from the menu bar

Step 1 From menu bar, click **Edit → Find**. Alternatively, you can press the shortcut keys (CTRL + F).

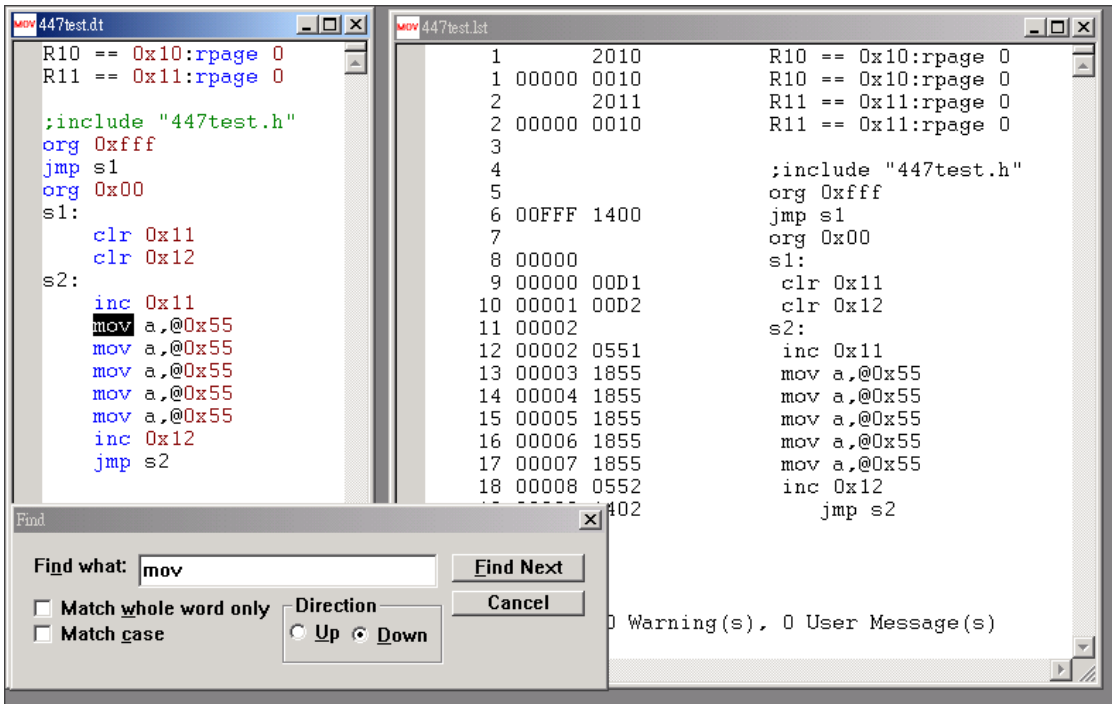
Step 2 From the resulting dialog, enter the string you want to find and its parameters.



Find what— Enter the string you want to search.
Match whole word only — Select this option to search string that matches the string as a whole word. For example, if you want to search for “abc”, the function will not pick out words that has additional characters (including spaces) before or after the defined string, such as “123abc” or “abced”.
Match case — Select this option to match the string lowercase and uppercase characters exactly as they appear.
Direction — **Up:** search toward the top of the document.
Down: search toward the bottom of the document.
Find Next —Click this button to continue searching for the next match. Matched string is highlighted.

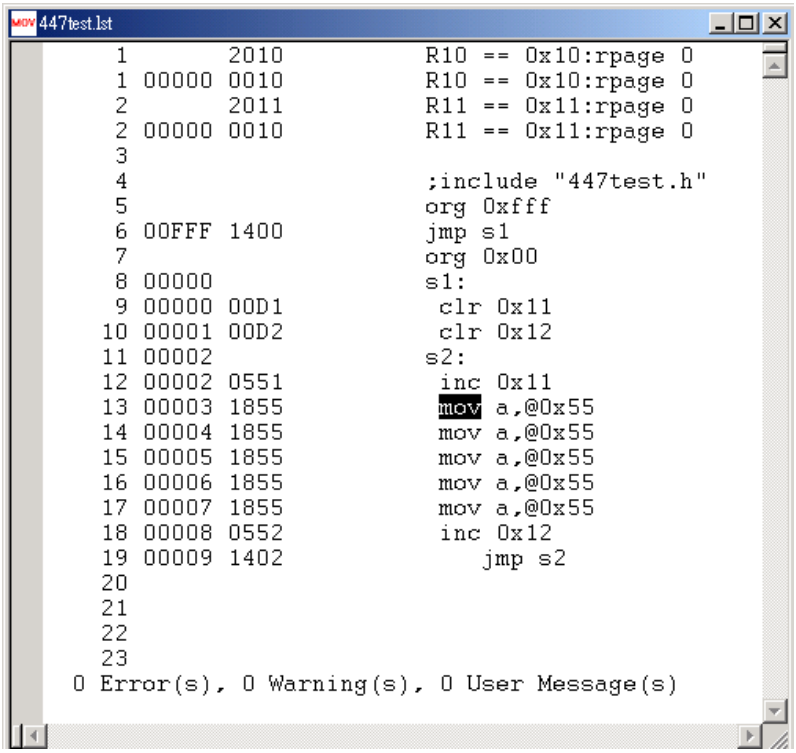
The following figure shows an example where **Find** function activate the location where a

match is found and highlights the search string.



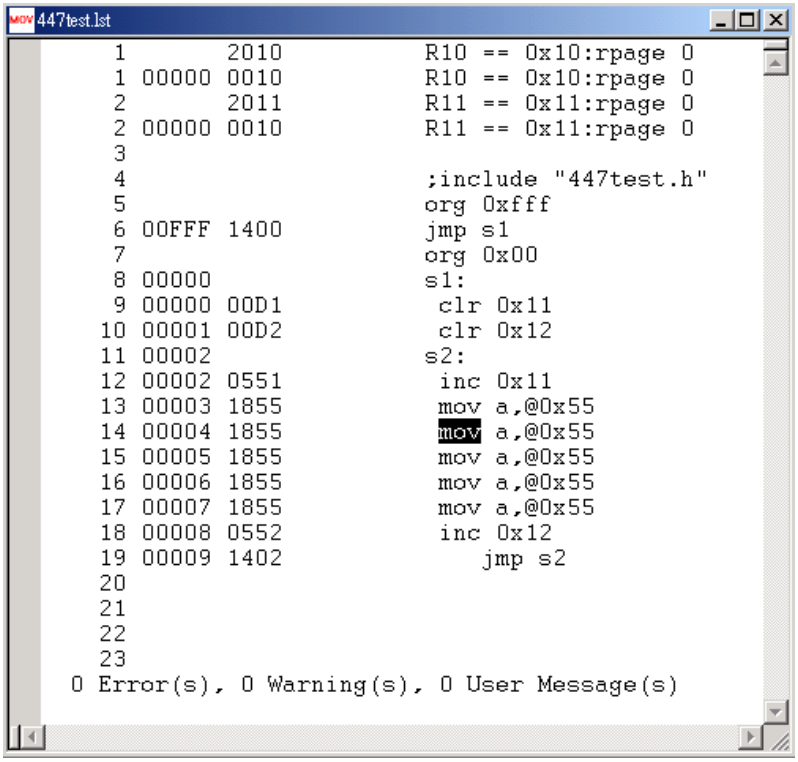
Method 2: Using the Shortcut keys

Step 1 Select the string you want to search from **Edit** window.



Step 2 To search upward, press the shortcut keys **CTRL + P**.

To search downward, press the shortcut keys **CTRL + N**.



2.3.3 View Menu

*Fig. 2-13 **View** Menu*

Project Show/hide **Project** window

Special Registers Show/hide **Special Register** window

General Registers (Bank) Show/hide **General Register (Bank)** window

CallStack Data Show/hide **CallStack** window

Data RAM Show/hide Data **RAM** window (if supported by the target chip)

LCD Data Show/hide **LCD Data** window (if supported by the target chip)

Output Show/hide **Output** window

Watch Show/hide **Watch** window

Assembly Code Only C Compiler use.

Show/hide Assembly Code in/from **Editor** window

After first dump to ICE (F3), press this button , then assembly code will display with C source

Toolbars Show/hide Standard, Build, or both toolbars

Status Bar Show/hide Status bar

Document Bar Show/hide Document bar

2.3.4 Project Menu

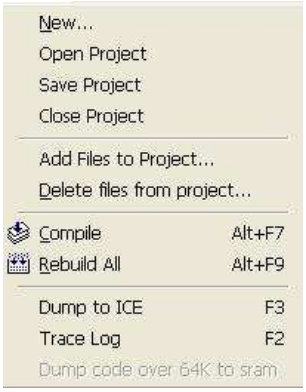


*Fig.2-14a **Project** Menu*

New… Create a new project

Open Open an existing project

Save Save the active project together with all related files

Close Close the active **Project** window

Add Files to Project… Add the existing source file into project

Delete Files from Project… Remove source file from project

Assemble (or Compile in C mode) To assemble (or Compile) the active file window. If errors occur in the assembler (or compiling) time, error messages will be shown in the output window; otherwise, " 0 errors, 0 warnings, 0 users" will be displayed.

Rebuild All To assemble (or compile) all files regardless of having been modified or not and link them to the opening project.

Dump to ICE Dump the program code to ICE

Trace Log To refer an available history only when the function of "Go", "Free Run", or "Go To Cursor" in the debug menu is executed. The maximum length of trace log is 8K words.
LPT connect: The last record is next instruction without execute.
USB connect: The last record is last instruction which has executed.

Dump code over 64K to sram Only for EM78815. To dump the page range's program code (over 64K) to the SRAM. The SRAM must plug in the hardware. Shown as Fig.2-14b.

Dump code over 64K to sram :“Start Page” is over 64, “End Page” is under 127.

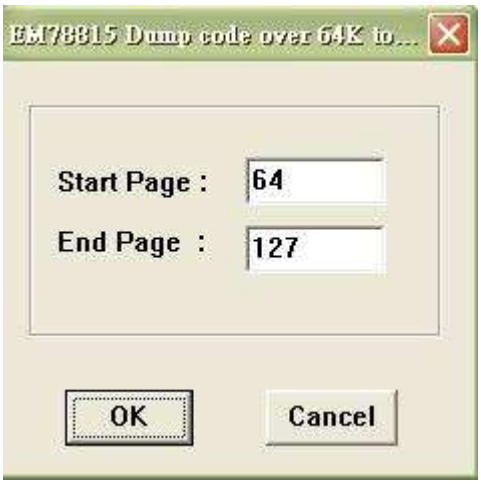


Fig.2-14b Dump code over 64K to sram

2.3.5 Debug Menu

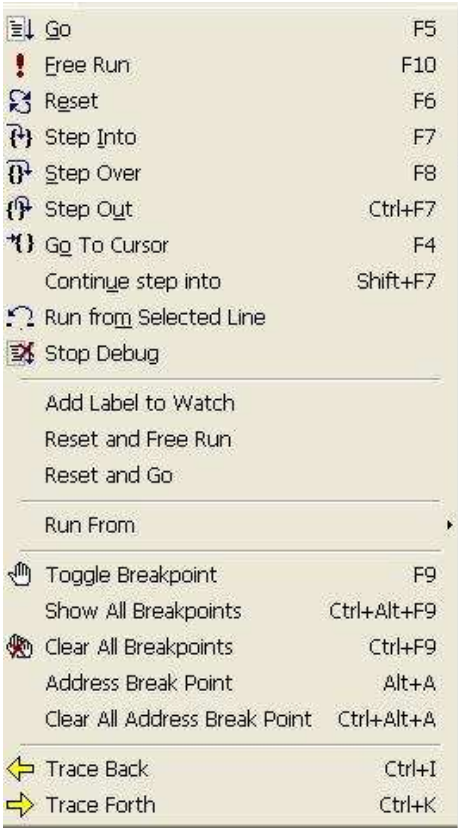


*Fig.2-15a **Debug** Menu*

Go Run program starting from the current program counter until a breakpoint is matched

Free Run Run program starting from the current program counter until the OK button of the "Stop Running" dialog is clicked

Reset Perform ICE reset (register contents are displayed with initial values)

Step Into Execute instructions step-by -step (with register contents updated simultaneously)

Step Over Execute instructions as "Step Into" (see above), but the CALL instruction will execute as "step over"

Step Out Exit CALL subroutines when execute "Step Into" into a CALL subroutines.

Go to Cursor Run program starting from the current program counter up to the location where the cursor is anchored (applies to ICE debug mode only)

Continue step into To perform the function of "Step into."

Run from Selected Line To apply this function, the start for next time will be located on caret.

Stop Debug Stop Free Run or Go without breakpoint.

Add Label to Watch To add or delete variables from the watch window. Shown as Fig.2-15b.

Reset and Free Run To reset the hardware and then execute the function of "Free run".

Reset and Go To reset the hardware and then execute the function of "Go"

Run From depicted in Fig.2-15c.

Toggle Breakpoint Set or remove a breakpoint

Show All Breakpoints	Show all breakpoints set-up data in the **Output** window
Clear All Breakpoints	Clear all breakpoints
Address Break Point	To define addresses to be breakpoints. Shown as Fig 2-15d.
Clear All Address Break Point	Clear all address breakpoint
Trace Back	After trace log is executed, “trace back” will be acted. The caret moves backward step by step from the last executed address to the address located before the current executed address.
Track Forth	After trace log is executed, “trace forth” will be acted. The caret moves forward step by step from the last executed address to the address which “trace back” ever went through.

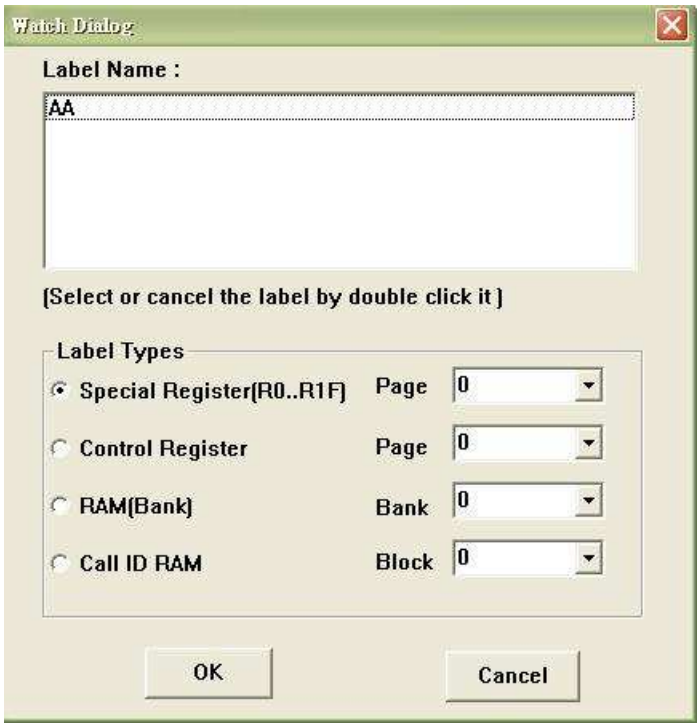


Fig.2-15b Add label to watch

Fig.2-15c The sub-function of “Run from”

(a) **Initial with 8K Step log**: To keep running program started from the initial address until breakpoints are hit. The last 8K steps of execution history are stored in the trace buffer.

(b) **Current PC with 8K Step log**: To keep running program started from the current program counter until breakpoints are hit. The last 8K steps of execution history are stored in the trace buffer.

(c) **Initial with 4K-4K log**: To keep running program started from the initial address until breakpoints are hit. The last 8K steps of execution history that the breakpoint address should be middle are stored in the trace buffer.

(d) **Current PC with 4K-4K log**: To keep running program started from the current program counter until breakpoints are hit. The last 8K steps of execution history that the breakpoint address should be middle are stored in the trace buffer.

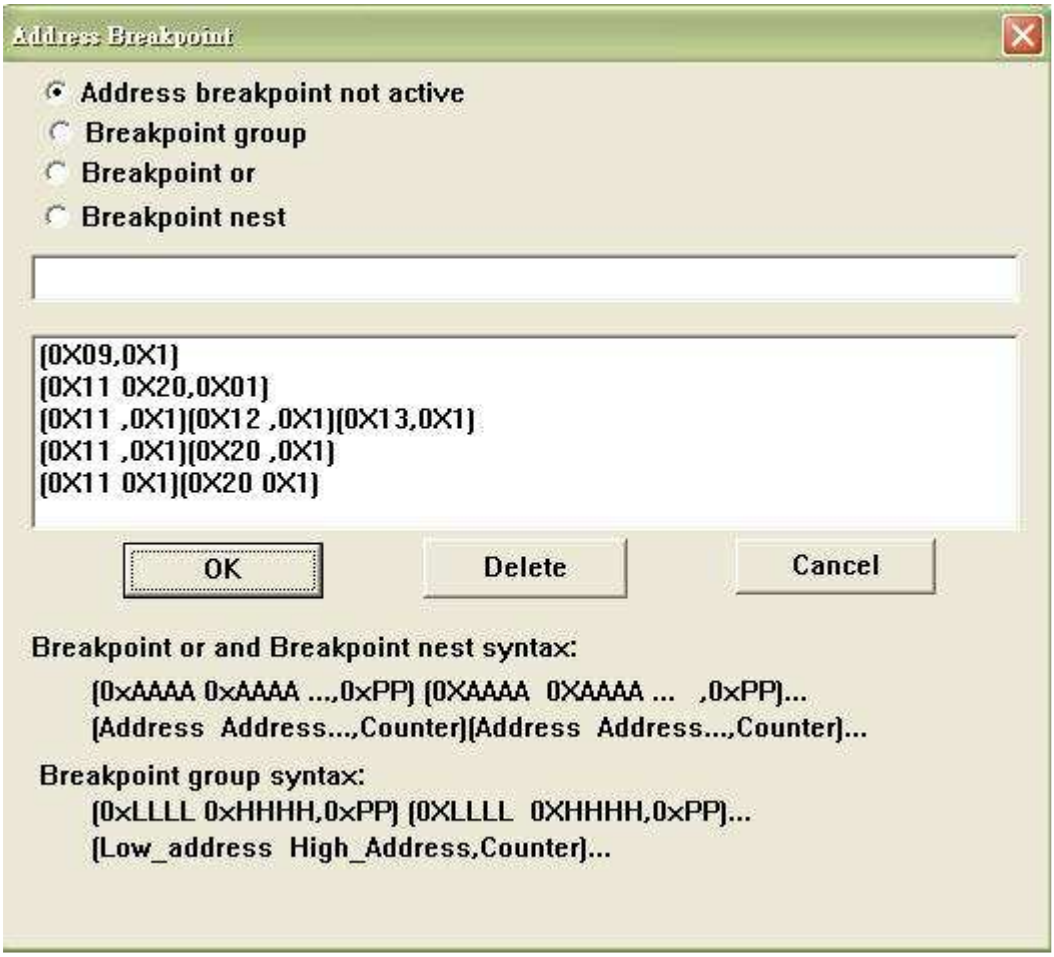


Fig. 2-15d: Address breakpoint

The breakpoint method in this dialog is the address breakpoint which is contradictory to the source level breakpoint; namely, the source level breakpoint will be inefficient if you use address breakpoint at the same time. Therefore, if you choose “Address breakpoint not active”, the method of the source level breakpoint will be effective.

Address breakpoints include three types: Group, OR and Nest, which are contradictory to each other. However, the set-up syntax for OR and Nest is the same, but it is different from Group's.

(a) **Breakpoint Group**: There are 63 groups at most and each group has start address, end address and pass count. When any instruction is executed between start address and end address, pass count will be deducted 1. As the pass count is equal to 0, the executing program is stopped at once. All of groups are independent of each other.

(b) **Breakpoint OR**: 63 groups are at most. Each group is composed of several addresses and a pass count. When any address is executed inside group, pass count will deduct 1. When the pass count is equal to 0, then a breakpoint is occurred. All of groups are independent of each other.

(c) **Breakpoint Nest**: Assign some address location as groups and specify those groups as a nest of breakpoint. The outer (the later specified) group must be first satisfied, then the inner group will take effect.

Breakpoint nest and breakpoints of program line are exclusive. In other words, if breakpoint nest is active, then breakpoints of program line are inactive. On the contrary, if breakpoint nest is inactive, then breakpoints of program line are active.

How to set breakpoint nest?

Step 1: Check the checkbox of "Breakpoint nest".

Step 2: Input breakpoint address. If address breakpoint is for (0x10 0x20 0x30, 0x55) (0x15 0x100 0x170, 0x10), address location 0x10, 0x20, 0x30 are assigned to the same group (Group 1), a breakpoint is attached to this group, the associated pass count of this breakpoint is 0x55. Address location 0x15, 0x100, 0x170 are assigned to another group (Group 2), a breakpoint is attached to this group, the associated pass count of this breakpoint is 0x10.

Step 3: Press [OK] button.

Step 4: To execute "Go (F5)" comment.

Step 5: If the Group 2 must be satisfied (pass count decrements to 0) then the Group 1 will take effect (decrease its pass count on meeting the breakpoint condition). As soon as all the groups are satisfied, the execution is stopped at the breakpoint.

Supplement

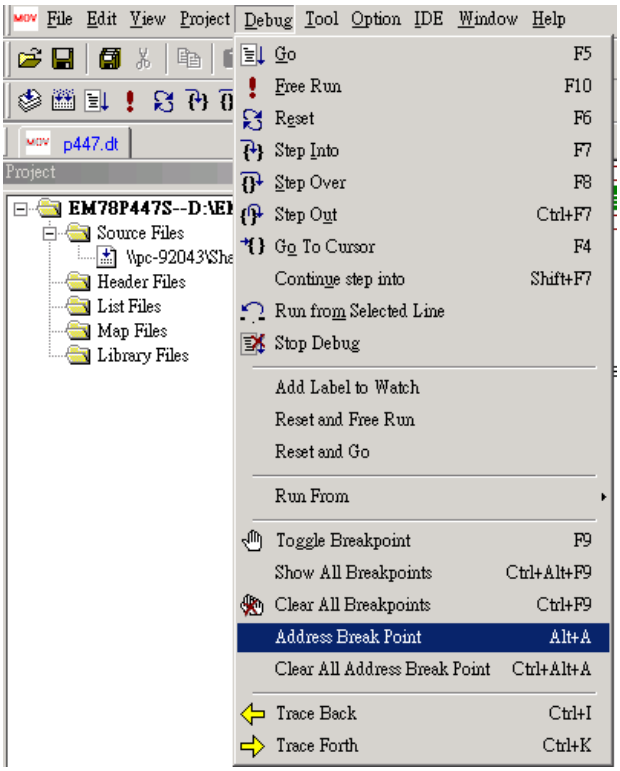
Supplementary information for address breakpoint.

The **Address Break Point** command allows you to directly set a more complex breakpoints setup which can terminate at an address location.

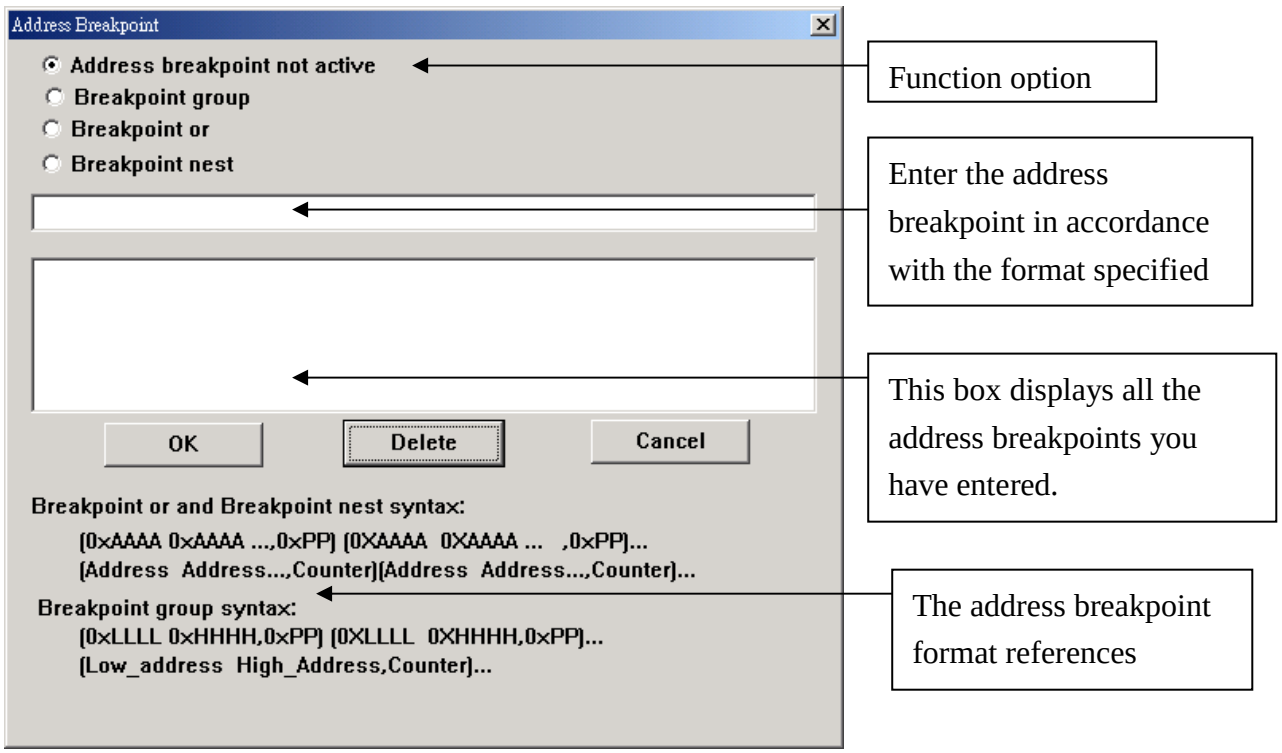
Setting an Address Breakpoint

Step 1 Open the **Address Breakpoint** dialog using one of the following methods**:**

- From the menu bar, click **Debug → Address Break Point**.
- Press the shortcut keys **ALT** + **A**.



Step 2 The **Address Breakpoint** dialog displays as shown in the following figure.



Step 3 Select an address breakpoint option (**Breakpoint group**, **Breakpoint or**, or **Breakpoint nest**). The following explains each of the options**:**

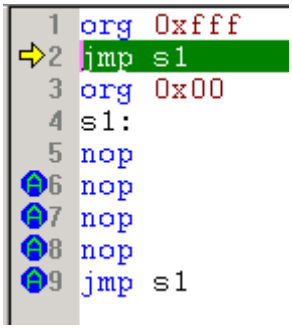
Breakpoint group—Select this option to set a group of breakpoints in a range of address locations with a counter number.

a. Select the **Breakpoint group** option.

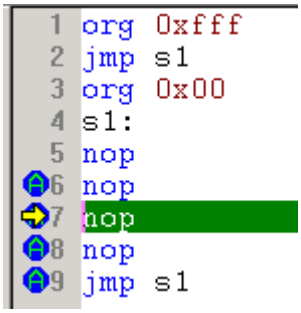
b. Specify a breakpoint group consisting of a start address, end address, and a breakpoint counter in the text box. A breakpoint will be set to all addresses within the start and end address range. The figure example below shows “(0X01 0X04, 0X01)” which means breakpoints are set to all addresses between “0X01” and “0x04” and break counter is set to “1” (thereby allowing a termination break on one passing).

c. Click **OK** to save the settings. You should see the group breakpoints indicated in the **Editor** window or as shown in the figure example below.

The following shows an example.



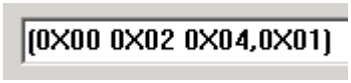
d)Click **Go** command to see the steps and result of the code execution.



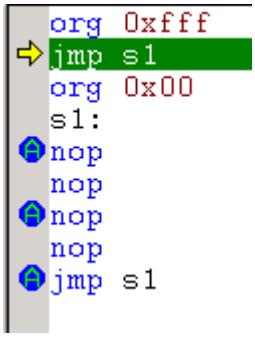
Breakpoint or—Select this option to set breakpoints at one or more address locations with a counter number.

a. Select the Breakpoint or option.

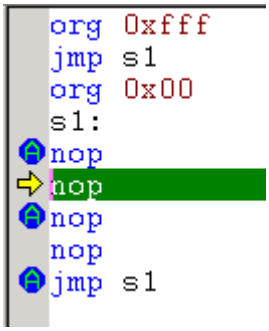
b. Enter the breakpoint address(es) and counter in the format “breakpointAddress1 breakpointAddress2 …, counter”. This means that a breakpoint is set at locations 0x01, 0x02, and 0x04 with the counter set to “1”.



c. Click **OK** to save the settings. You should see the group breakpoints indicated in the **Editor** window or as shown in the figure example below.



d. Click **Go** command to see the steps and result of the code execution.



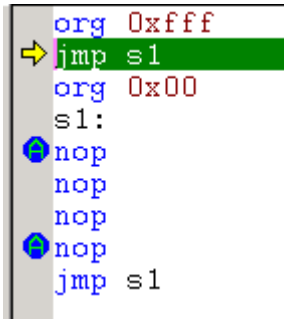
Breakpoint nest

Select this option to set nested breakpoints consisting of groups of address locations with each group assigned with a counter number.

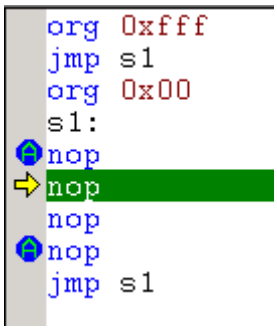
a. Select the Breakpoint nest option.

b. Enter the breakpoint address(es) and a counter in the format of – "***breakpointAddress1 breakpointAddress2 …, counter***". The following figure shows an example with "(0X00, 0X01) (0X03, 0X02)". This means that the program will run address 0x03 twice, run address 0x00 only once and then terminate.

c. Click **OK** to save the settings. You should see the breakpoints indicated in the **Editor** window or as shown in the figure example below.



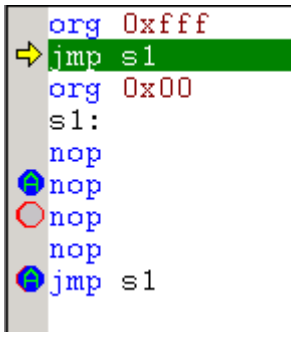
d. Click **Go** command to see the steps and result of the code execution.



Note 1: You can only set breakpoints after performing dump action. If you try to set breakpoint without dumping, a pop-up message will display to prompt you to execute dump action..

Note Breakpoints (solid blue dot) created through the **Address Break Point** command (as

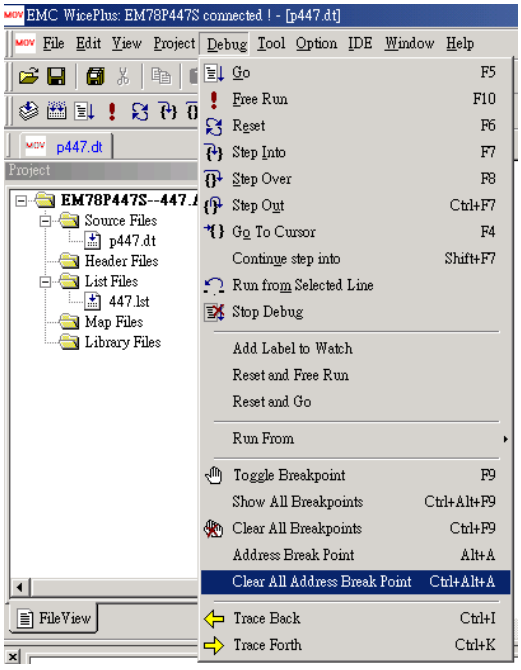
explained above) are independent from the breakpoints (solid red dot) created individually by double-clicking on the pointed address (or by pressing **F9**). Since they do not conflict with each other, the blue and red address breakpoints can co-exist at the same time and at the same address location. However, the blue breakpoints has the priority over red ones when **Go** command is executed. Hence, any existing solid red dot breakpoints are ignored.



Note 3: As the Address Breakpoints dialog created blue breakpoints has the priority over the individually-created red breakpoints, the solid red dots turns into hollow red circle when both types of breakpoints occupy the same address. Once the blue address breakpoints are cleared, the hollow red circle becomes solid again. The red address breakpoints are applicable only when the blue ones are all cleared.

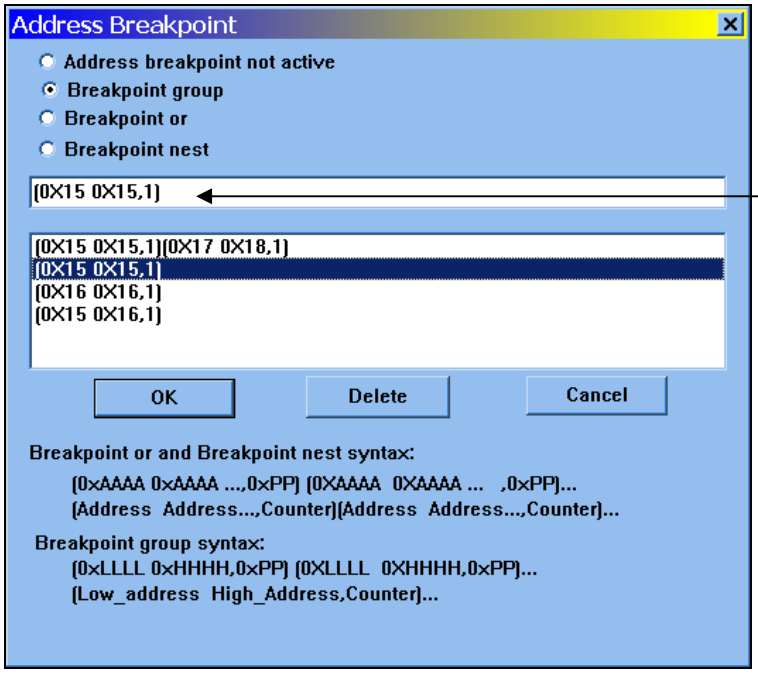
Note 4: To clear all blue breakpoints, do either of the following**:**

- From the menu, click Debug → Clear All Address Break Point
- Press the shortcut keys CTRL + ATL + A



5. To clear individual or a range of blue breakpoints, do the following:

a) From the menu, click Debug → Address Break Point (or press Alt+A)



You can also enter the specific or range of address breakpoint(s) you want to delete here.

b) From the resulting **Address Breakpoint** dialog which displays all the existing blue breakpoints, select the specific or range of breakpoint(s) to delete from the dialog.

c) Click **Delete** to remove breakpoint.

2.3.6 Tool Menu

*Fig. 2-16a **Tool** Menu*

Connect	Define printer port connection with ICE (default is 378H)
Check ICE Memory	Check available memory from ICE
Get Checksum from Project	Obtain checksum from the compiled program
Piggy back MIX format	To create a MIX file used to write to EEPROM on a piggyback
Piggy back Hi Lo format	To create a couple of files with extension names of Hi and Lo, which are used to write to EEPROM on a piggyback
Clear all output mapping line	To clear program mapping lines from output window
Get Code=1FFF size	Get free code size
Get TBL code position	Get TBL code position
Piggy back MIX2 format	For EM78813 & EM78815. If use original 8 bit to fit 16 bit piggy back, high byte in 8 bit piggy back will become low byte in 16 bit piggy back. low byte in 8 bit piggy back will become high byte in 16 bit piggy back.
Compute Execution Time	Compute the execution time between two breakpoints. Shown as Fig.2-16b.
Move data from file to sram	Only for EM78815 use. Shown as Fig.2-16c.
Speed Up Debug	Only for C Compiler use
Generate Approval Form…	This function is not yet supported

a.Compute Execution Time :

How to compute execution instruction time of ICE?

Step 1: To select frequency. To see Fig 2.16b.
Step 2: To execute TraceBuffer (F2) of WicePlus. The result is in message window.
Step 3: The result of execution times is in information window.

Fig. 2-16b Set Frequency

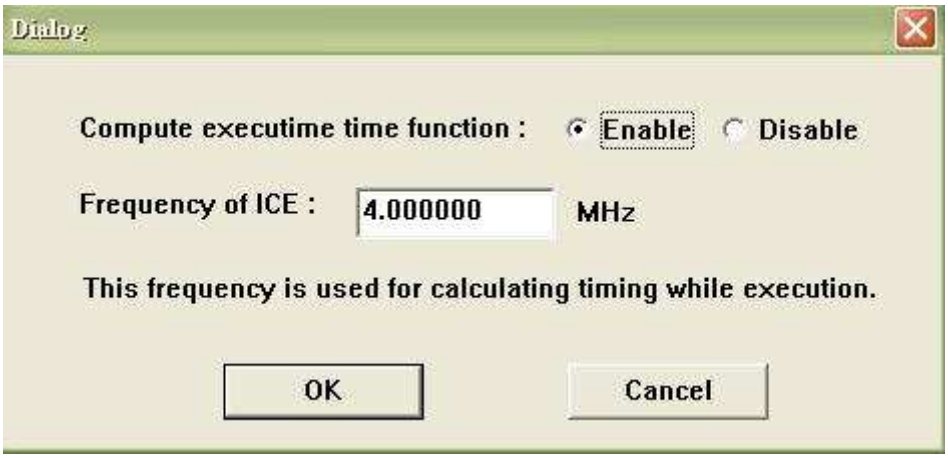


Fig. 2-16b1 Set Frequency

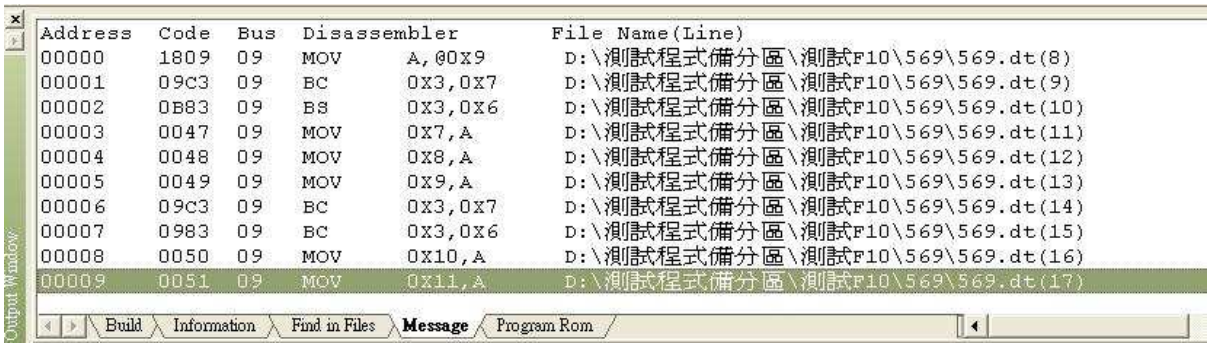

Address Code Bus Disassembler File Name(Line)
00000 1809 09 MOV A,@0X9 D:\測試程式備分區\測試F10\569\569.dt(8)
00001 09C3 09 BC 0X3,0X7 D:\測試程式備分區\測試F10\569\569.dt(9)
00002 0B83 09 BS 0X3,0X6 D:\測試程式備分區\測試F10\569\569.dt(10)
00003 0047 09 MOV 0X7,A D:\測試程式備分區\測試F10\569\569.dt(11)
00004 0048 09 MOV 0X8,A D:\測試程式備分區\測試F10\569\569.dt(12)
00005 0049 09 MOV 0X9,A D:\測試程式備分區\測試F10\569\569.dt(13)
00006 09C3 09 BC 0X3,0X7 D:\測試程式備分區\測試F10\569\569.dt(14)
00007 0983 09 BC 0X3,0X6 D:\測試程式備分區\測試F10\569\569.dt(15)
00008 0050 09 MOV 0X10,A D:\測試程式備分區\測試F10\569\569.dt(16)
00009 0051 09 MOV 0X11,A D:\測試程式備分區\測試F10\569\569.dt(17)
Output Window
Build
Information
Find in Files
Message
Program Rom

Fig. 2-16b2 The result of trace buffer in Message sub-window

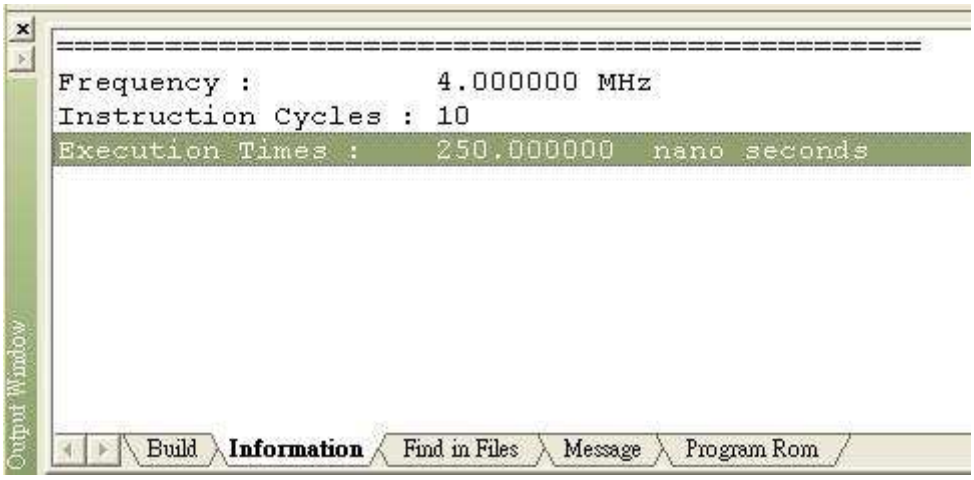


Fig. 2-16b3 The result of execution times in Information sub-window

b. Move data from file to sram (Only foe EM78815 use)

We support some functions in the new version WicePlus that is consistent with external 512K byte SRAM board. Our purpose is to use external SRAM rather than flash. The followings are the features of these new functions before detail descriptions.

1. The user can move data from binary file to external SRAM;
2. The user can use external SRAM to develop program when program's size over 64K word;

Following we'll introduce the details. Please remember all we mention. We suppose your connection is OK , ready to debug your program and in your program you have to access data from external memory. **Now we replace external flash ram with external SRAM in developing process.** There are two important parts in our description: **Move data from binary file to external SRAM** and **Dump code to external SRAM**.

Move your data from binary file to external SRAM:

1. **Reset (F6). This is very import before you move data from binary file to external SRAM**. But not always to press reset menu before moving data from binary file to external SRAM. You have to do this when you just connect ICE and do not reset (F6) or not debug your program yet. If you forget to do it, what moving data from file to SRAM will fail. We mean the user have to initialize the environment of ICE and Wice before moving binary data into external SRAM.
2. Press Move data from file to SRAM in Tool menu.

3. Select the binary file you want to operate and set the data begin address and length.

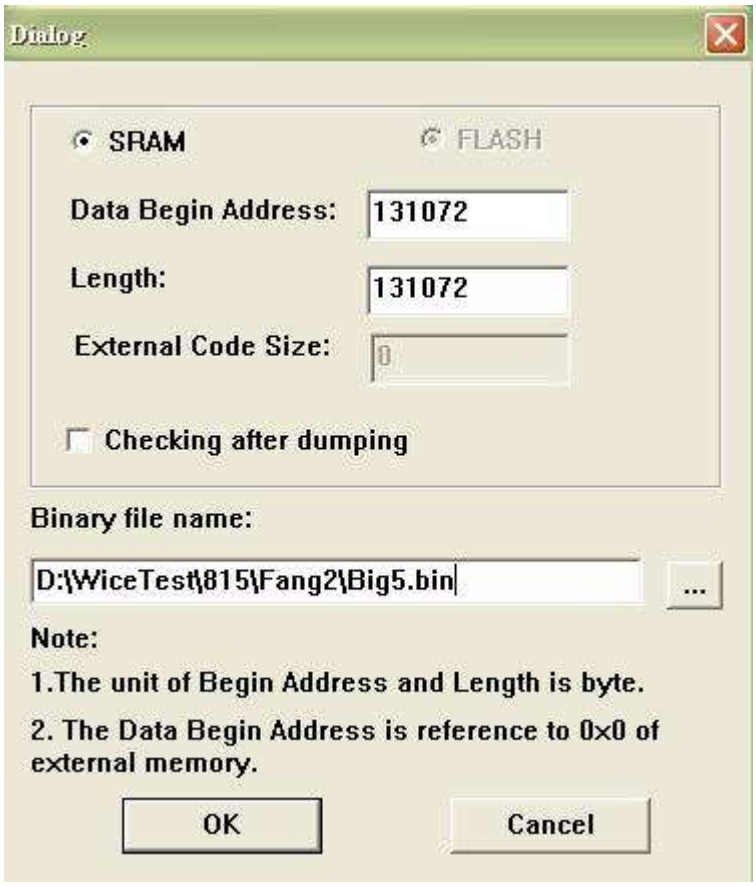


Fig. 2-16c **Move data from file to sram**

Please read the notes before setting the Data Begin Address and Length. After type the correct number and select the binary file, press OK button to move your data. You can check Checking after dumping box to verify data after finishing the process. If it meets one of the bytes is error, it will stop verifying.

External-Code-Size Edit will tell you how many bytes your program code over 64K word is. **So let your Data Begin Address bigger than External-Code-Size.**

> The maximum of the Data Begin Address is 524,288(512K byte) and the Length 262,144(256K byte). So if the data you want to move from binary file to external SRAM is over 262,144 byte (256K byte), you have to divide your binary file into two files and move your data two times.

For example, now we have a 512K byte external SRAM, a binary file, named Big5.bin, and its size is 393,216 byte (384K byte). We want to move all content of the Big5.bin into external SRAM from the Data Begin address 131,072.

So, we have to divide the binary file into two files: Big51.bin and the Big52.bin. Their size is 262,144 byte (256K byte) byte and 131,072 byte (128K byte).

Then we set Data Begin Address as 131,072, Length as 262,144, Binary file name as Big51.bin to move data into external SRAM. The data will be located from address 131,072 to address 393,215.

After the previous process is finished, we set Data Begin Address as 393,216, Length as 131,072, Binary file name as Big52.bin to move data into external SRAM again. The data will be located from address 393,072 to address 524,287 after finishing the process.

You just have to do the moving data process only one time if you don't power off ICE or change any data in your binary file. We suggest the user just do it one time one day.

Dump code to external SRAM:

If your program code is over 64K word, now you press Dump to ICE (F3) to dump program code to ICE. After process and checking code under 64K word, your code over 64K word will continue to be dump to external SRAM. It will take about 45 seconds to dump full of this part. How many seconds it takes to dump external SRAM depends on your code size over 64K word. You could enable the external-code-checking box in Environment setting Dialog, Option menu. **We recommend you to enable external-code-checking box in first dumping and disable it after first dumping to save another about 40 seconds.**

If you program code is less than 64K word, it will not dump to external SRAM.
Shown as Fig.2-17e.

2.3.7 Option Menu

Fig. 2-17a ***Option*** *Menu*

ICE Code Setting	Set code option for the selected microcontroller
Font…	Define font for **Editor** windows (fonts for other windows are fixed)
Debug option setting	To set debugger variables options. Shown as Fig.2-17b.
Dump ASPCM	Dump to Data ROM.
Variable Radix	Select between decimal or hex option
Accelerate Reading Registers	Only for USB ICE. When frequency over 1.6MHz, read registers quickly.
View Setting	To set WicePlus view variable, Shown as Fig.2-17d.
Enviroment setting	To set WicePlus environment variable, for example, whether list file is created or not, whether map file is created or not and the number of editor window. Shown as Fig.2-17d.
Customize…	Customize toolbars, menus, and accelerators
TCC2	Select Tiny C Compiler version 2. Uncheck is TCC1

a. **Debug option setting**: To set debugger variables options.The option is divided into the following four blocks shown in Fig.2-17b:

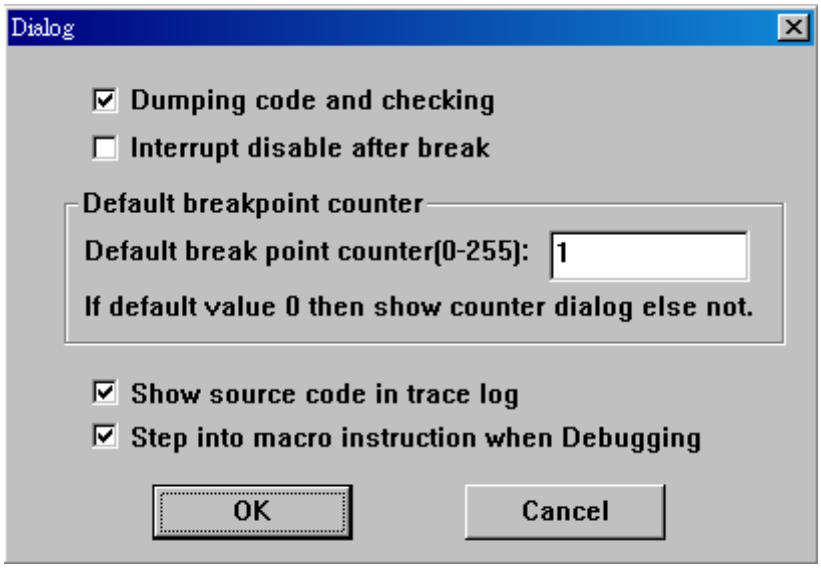


Fig.2-17b Debugger option setting

(a) Dumping codes and checking: WicePlus will check the equipped memory of the hardware before dumping codes.

(b) Interrupt disabled after program stopped: Interrupt is disabled as a breakpoint occurs. It is used to avoid any interrupt occurred during updating screen, because TCC, COUNTER1 and COUNTER2 would keep working even program is stopped. Therefore, the disabled interrupt must be active; otherwise users cannot debug the program.

(c) Default breakpoint counter: Please take Chapter 5 -- Debugger: a source level debug-- as a reference.

(d) Show trace log in source program: In the output window, the contents of trace log are shown by default setting in the contents of disassembler. If this function is chosen, address source level breakpoint produced by the trace log address will appear in the output window.

(e) Setp into macro instruction when Debugging: When this option check box is enabled, click StepInto from the Debug menu to run the program instructions including the macro step-by-step. To run the macro code in the background, click StepOver (F7). As a new feature, you can disable check box to run the macro in the background, regardless of whether you click StepInto (F7) or StepOver (F8). Please remember that the instruction line that is currently being executed will stop at the address location of the macro (shaded in green). This location will point to the first line of the macro code. click StepInto (F7) to execute the first macro instruction and then its stop at the second macro instruction (shaded in green again).

Example:

```
1. test macro
2. nop
3. nop
4. endm

5. org 0x0
6. jmp main
7. main:
8. nop
9. test
10. nop
11. jmp $
```

Per above code example, if the program execution stops at Line 8 (shaded in green on screen), click StepInto (F7) to execute Line 8 and then execution stops at Line 9 (note that in previous WicePlus version, it stops at Line 2 of the macro).

As Line 9 is actually linked to Line 2 address, clicking StepInto (F7) will execute the code at Line 2 and stops at Line 3. If you click StepOver (F8), the program executes the complete macro instructions (Line 2 & Line 3) and stops at Line 10.

b. Accelerate Reading Registers

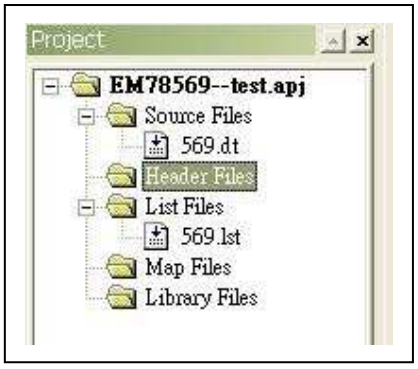
Accelerate Reading Registers
Accelerate Reading Registers
Enable [instruction clock >= 2 MIPS]
Disable [instruction clock < 2 MIPS]
OK

Fig.2-17c Accelerate Reading Registers

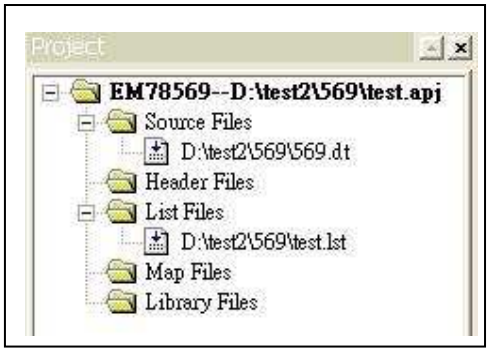
When you use USB ICE and the frequency is over 2 MHz, you can enable this capability. But it is not 100% stable. If you see some abnormal appearances, try to disable this option.
Note: If you choose IRC when you set the code option, WicePlus may enable this option automatically according to the chosen frequency.

c. **View *setting***: To set WicePlus view variable,

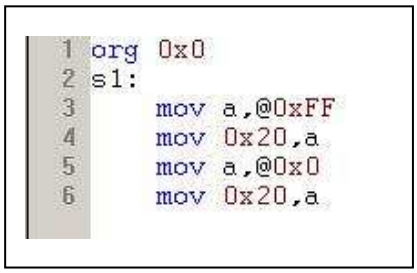
Fig.2-17d WicePlus environment



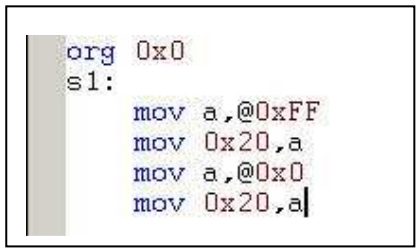
(a)File name



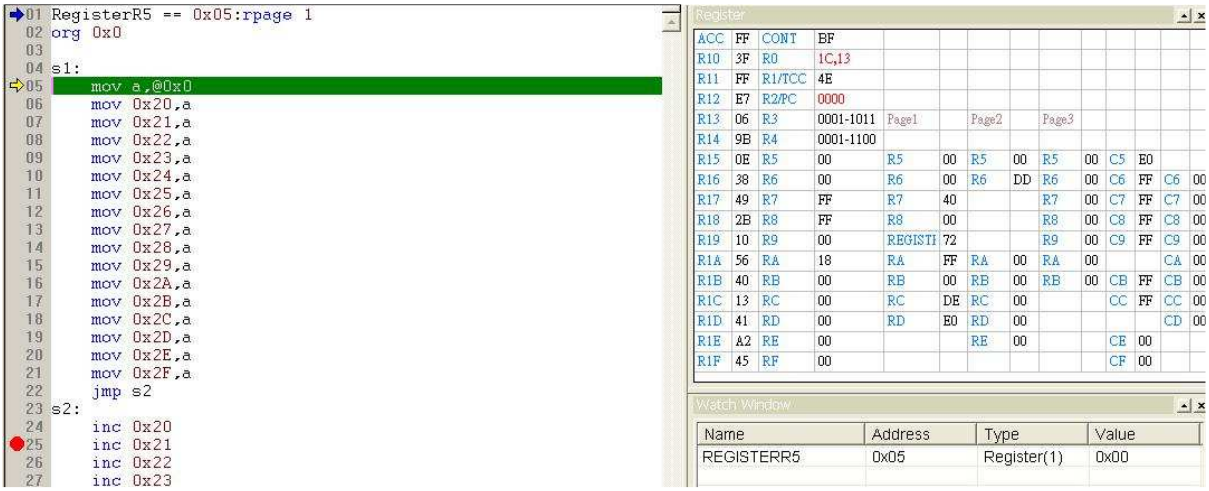
File name with path



(b)Column On



Column Off



(c) Tab Size: customize tab size.

(d)Add defined label to watch automatic: (Enable after Dump)

Checked: To add definite variables with rpage/rbank/iopage in Watch Window automatic.

(e)Show "Unreferenced variable": When build assemble code inspects the variable whether to have the use.

(f)Show defined label in Register Window: (Enable after Dump)

Checked: Register's names are label name in Register Window.

If the label name's length over 6. Register name display front six char, and tooltip display complete name.

Unchecked: Register's names are initial name in Register Window.

(g) Show program Rom: To show the program code at **Program Rom**

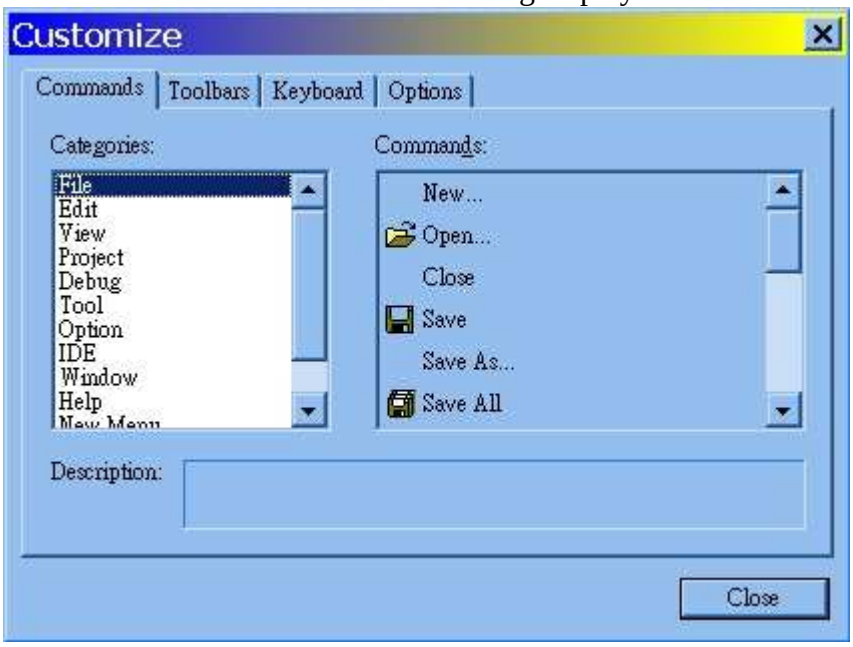
sub-window of **Output** window.

d. **Environment setting**: To set WicePlus environment variable, for example, whether list file is created or not, whether map file is created or not and the number of editor window.

Fig.2-17e Setting environment

(a) **Create List file**: If chosen, the LIST file is created after a related project is assembled. The LIST file includes line number, address, program code and source file.
(b) **Create MAP file**: If chosen, the related LIST file is created after a project is linked. The MAP file includes public symbol name and address.
(c) **Recent file list contains**: The number of recent closed file name can be saved in WICE. The maximum number of editor window is 10.
(d) **Recent project list contains**: The number of recent closed project name can be saved in WICE. The maximum number of editor window is 10.
(e) **Auto Dump Over 64K**: Only for EM78815. If chosen, dump all program code to the hardware.
(f) **External Code checking**: Only for EM78815. If chosen, to check the action of dumping program (or data) to external rom.
(g) **Show Trace Log Under 64K**: Only for EM78815. If chosen, to refer inside 64K program code only when the function of “Go”, “Free Run”, or “Go To Cursor” in the debug menu is executed. The maximum length of trace log is 8K words.

e. **Customize…:** The **Customize** dialog displays with four tabs as shown below.

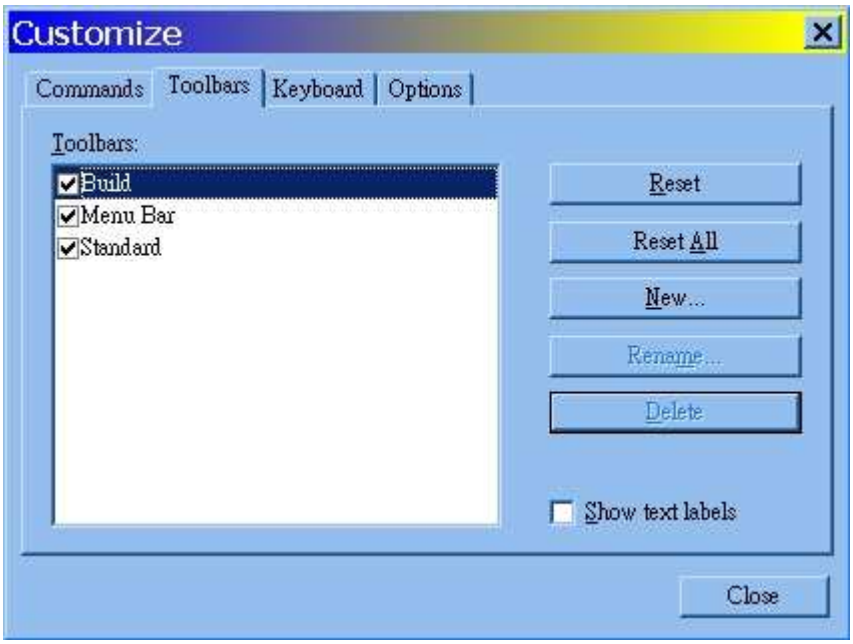


The following describes the function of each tab**:**

(a) Commands tab :

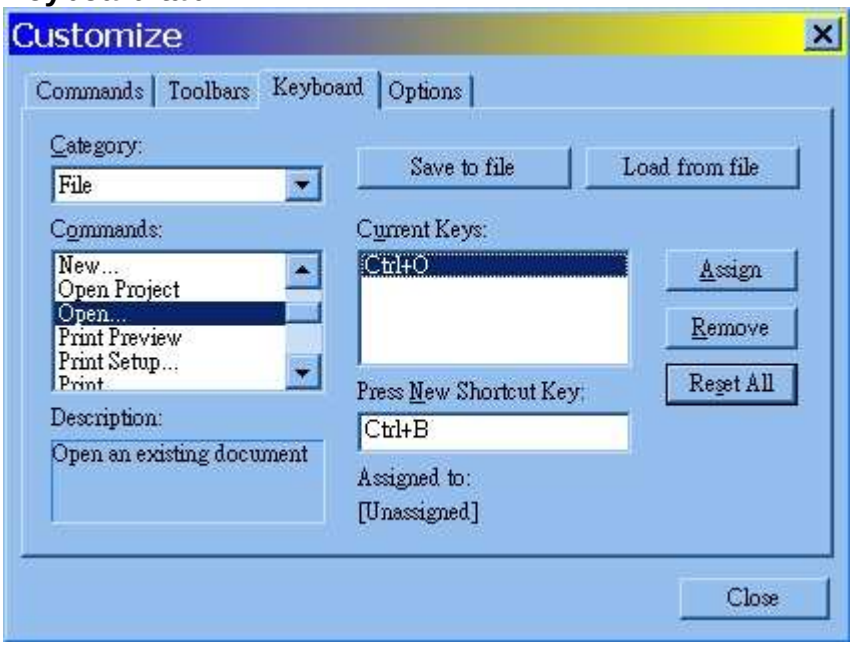
Select this tab to display all available commands under a selected category. Then drag and drop a command into toolbars, menu bar, or into a drop-down command menu (from menu bar). To restore default settings, go to **Toolbar** tab and click **Reset All**.

(b) **Toolbars tab**



The **Toolbars** tab allows you to enable/disable the **Build** and **Standard** toolbars but not the **Menu Bar**. You can however, **Reset/Reset All** all toolbars to restore the default settings of a particular or all toolbars. Furthermore, you can create your own new unique toolbar which you can later **Rename** or **Delete** from this tab. You can also add/remove text labels to the toolbar buttons by clicking on the **Show text labels** check box.

(c) Keyboard tab



The **Keyboard** tab allows you to create/remove shortcut keys for the commands of a selected command category. The procedure is explained below:

A. Creating a shortcut key**:**

After selecting a **Category** and **Commands** option from their respective boxes, enter your custom shortcut key into the **Press New Shortcut Key** box. WicePlus will auto–detect whether the new shortcut key has already been assigned or not. If it has already been assigned, the pertinent command name (with which the shortcut key is currently assigned) will display under **Assigned to:** field and you need to directly enter another shortcut key. Otherwise, *[Unassigned]* will display. Then click **Assign** to apply.

If the **Assigned to:** field displays "*????*", it means the shortcut key you just entered is reserved for WicePlus internal use and is not user definable.

B. Removing an existing shortcut key**:**

After selecting a **Category** and **Commands** option from their respective boxes, the corresponding command shortcut key (if available) will appear in the **Current Keys** box. Select the shortcut key and click **Remove** to delete.

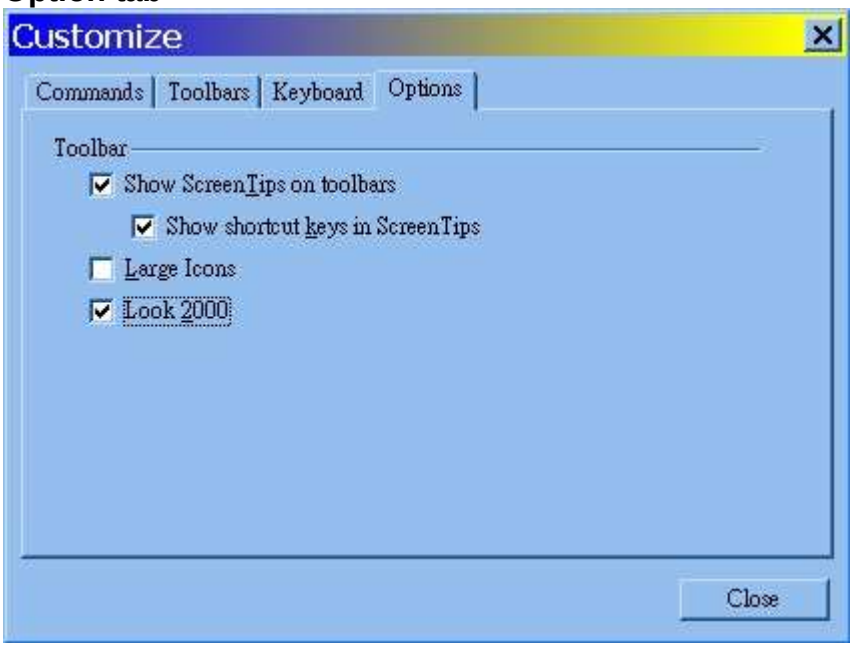
C. Restoring all shortcut keys to their default settings**:**

Click **Reset All** to reset all command shortcut keys to the WicePlus default settings.

D. Saving/Loading settings:

To save your custom shortcut keys for future use, e.g., when upgrading WicePlus to a new version, click **Save to File** to store the settings to a file (with a ".KEY" file extension). You can simply click **Load from file** to re-apply your custom shortcut keys into WicePlus.

(d) Option tab



Use the **Option** tab to set the size of the toolbar buttons and specify whether to display screen tips and shortcut keys (where applicable) when pointing at the button. You can also set to display the WicePlus sub-windows to look like that of Windows 2000.

2.3.8 IDE Menu

*Fig. 2-18 **IDE** Menu*

C Select C language editing

Assembly Select Assembly language editing

2.3.9 Window Menu

*Fig. 2-19 **Window** Menu*

New Window Open a new (or split) **Editor** window

Cascade Rearrange all **Editor** window active files so that they appear overlapping in sequence with their respective title bar fully visible

Tile Vertical Rearrange all opened **Editor** windows vertically

Tile Horizontal Rearrange all opened **Editor** windows horizontally

Arrange Icons Arrange all opened file filenames in a single line formation (minimized into multiple file icons) at the bottom of the **Editor** window.

> **NOTE**
> *This command is effective only after clicking the Minimized button ( ) at the right end of the WicePlus window Menu bar.*

Close All Close all opened files

Windows… Show Select Window Dialog

2.3.10 Help Menu

*Fig. 2-20a **Help** Menu*

User Manual **User Manual**

Example for C Only for C Compiler

Content… Open the WicePlus User's Manual

Index… Index from WicePlus User's Manual

Search… Search from WicePlus User's Manual

Check New Version… Check new version of WicePlus from EMC.

ELAN IDE
New version is announced.
Please go to http://www.emc.com.tw/twn/products.asp?np_id=1&type_flag=ST
to download the latest version!
Do you want to link to EMC WicePlus web site right now?
是(Y)
否(N)

Fig. 2-20b check new version

About… Shows the current version of WicePlus program and other information including a “read me” file on recent changes of the WicePlus

Fig. 2-20c **About** Dialog

Register ELAN To register to ELAN

2.4 Toolbar

Fig. 2-21a WicePlus Main Window (Standard) Toolbar

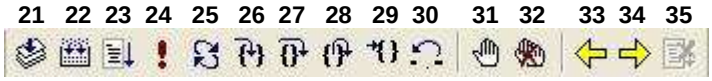


Fig. 2-21b WicePlus Main Window (Build) Toolbar

2.4.1 Toolbar Icons and Functions

Corresponding hot key is enclosed in parenthesis**:**

1 **Open:** open an existing file (Ctrl + O)

2 **Save:** save current active document (Ctrl + S)

3 **Save All**: save all document

4 **Cut:** remove the selected string to clipboard (Shift + Del)

5 **Copy:** copy the selected string to clipboard (Ctrl + C)

6 **Paste:** paste the string from clipboard (Ctrl + V)

7 **Undo:** cancel the last editing action (Alt + Backspace)

8 **Redo:** cancel the last "undo" (restore the "undone" editing action)

9 **Open/Hide Workspace:** display/hide toggle for **Project** window

10 **Open/Hide Output:** display/hide toggle for **Output** window

11 **Find:** find string from within entire active file (Ctrl + F)

12 **Find Down:** find string from cursor position toward the end of file

13 **Find Up:** find string from cursor position toward the beginning of file

14 **Find from Files:** find string from inactive files

15 **Bookmark:** bookmark the line at cursor position

16 **Jump Down to Bookmark:** jump to next bookmark from cursor position toward the end of file

17 **Jump Up to Bookmark:** jump to next bookmark from cursor position toward the beginning of file

18 **Clear Bookmarks:** clear all bookmarks

19 **Print:** print the active file (Ctrl + P)

20 **About:** about assemble version and other information

21 **Assemble (or Compile in C mode):** assemble (or Compile)the active file in the **Editor** window (Alt + F7)

22 **Rebuild All:** assemble (or compile in C mode)all files in the project and link object files(Alt + F9)

23 **Go:** auto dump and execute program with the effect of the breakpoints (F5)

24 **Free Run:** auto dump and execute program with the breakpoints ignored (F10)

25 **Reset:** reset the ICE (F6)

26 **Step Into:** auto dump and execute program step by step including subroutines (F7)

27 **Step Over:** auto dump and execute program step by step excluding the subroutines (F8)

28 **Step Out:** auto dump and execute program untilexit the subroutines (Ctrl + F7)

29 **Run to Cursor:** auto dump and execute program, then stop at the cursor position while ignoring the breakpoint (F4)

30 **Run from Selected Line:**To apply this function, the start for next time will be located on caret.

31 **Insert/Remove Breakpoint:** insert/remove toggle for breakpoints (F9)

32 **Remove All Breakpoints:** remove all breakpoints

33 **Trace Back**: After trace log is executed, “trace back” will be acted. The caret moves backward step by step from the last executed address to the address located before the current executed address.

34 **Trace Forth**: After trace log is executed, “trace forth” will be acted. The caret moves forward step by step from the last executed address to the address which “trace back” ever went through.

35 **Stop Debug:** Stop Free Run or Go without break point.

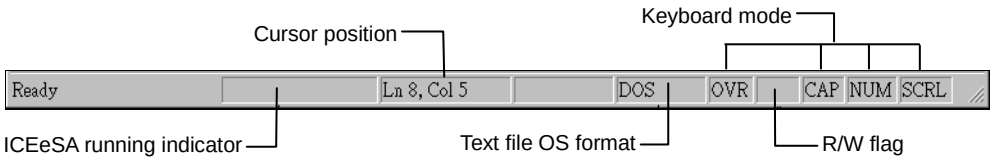
2.5 Document Bar

Fig. 2-22 WicePlus Main Window Document Bar

The **Document** bar displays the file icons representing each of the opened files in the **Editor** window. Click the icon of the pertinent file that you wish to activate (place in front of the **Editor** window to perform editing). Highlighted filename is the active file (function is similar with taskbar buttons under Windows).

To double cluck left button on the table in the **Document** bar to close active document.

2.6 Status Bar



A WicePlus running indicator will be shown in the **Status** bar while your project is being compiled.

The Cursor position indicates the cursor location within the text **Editor** window.

R/W flag indicates the active file Read/Write status. If Read only, “Read” will display, otherwise the field is empty.

Keyboard mode displays the status of the following keyboard keys**:**

- **Insert** key – OVR is dimmed when overtype mode is off, highlighted when on.
- **Caps Lock** key – CAP is dimmed when uppercase character mode is off, highlighted when on.
- **Num Lock** key – NUM is dimmed when the numeric keypad calculator mode is off, highlighted when on.
- **Scroll Lock** key – SCRL is dimmed when cursor control mode is off, highlighted when on.

2.7 Hot Key

- "Ctrl + Alt + Shift +G" ：Big5 to GB Chinese Code Converter
- "Ctrl + Alt + Shift +B" ：GB to Big5 Chinese Code Converter
- "Ctrl + Alt + Shift +Z" ：Undo

Converter documents with Chinese language.

Chapter 3
Getting Started

3.1 Hardware Power-up

With the E8-ICE properly connected to target board, PC, and power source, switch on ICE power and observe its red power LED lights up. If the target board derives its power from ICE, the yellow LED lights up as well.

Then launch your WicePlus IDE software when ICE and target board power-up is confirmed to function normally.

3.2 Starting the WicePlus Program

To start WicePlus Program, click on the WicePlus icon from desktop or from Windows Start menu. When starting from the Start menu, click Programs, then look for WicePlus group and click on WicePlus icon.

3.2.1 Connect Dialog

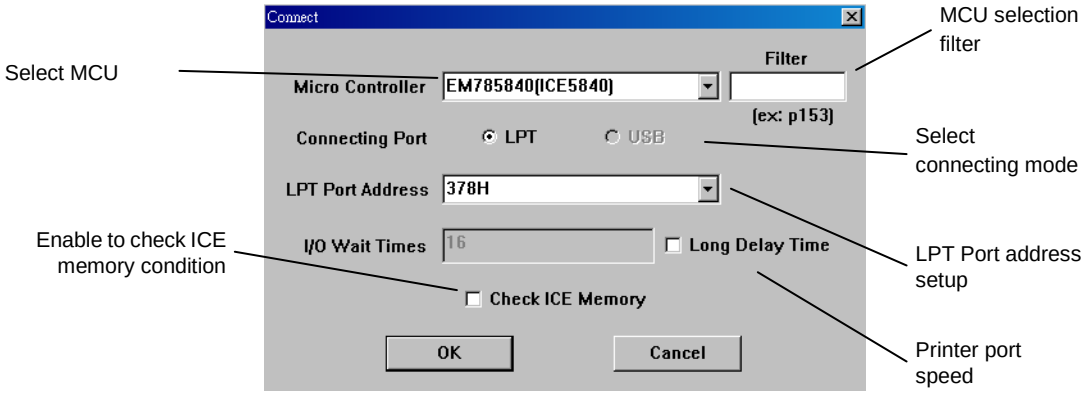


*Fig. 3-1 WicePlus Program **Connect** Dialog*

Once the program is started, the main window of the program will initially display the **Connect** dialog to prompt you to set the proper connection between your existing target microcontroller and connecting port.

Filter: Key in some desired IC's name, for example 159, Micro Controller combo box will just contain these IC's name with 159. It makes select IC easily. WicePlus version 2.6 or later supports this function.

Micro Controller: The device presented on the edit bar is one of the EM78XXX series. Users can also select others by clicking the downward triangle icon on the right side of the edit bar.

Connecting Port: You can choose LPT or USB for connecting port. If USB ICE is not detected, you cannot choose USB option. The option will be disabled. If the option is disabled and USB ICE has already been connected with PC. Please check the Device Manager of OS and try under steps:

- Turn off the ICE.
- Disconnect the USB cable.
- Turn on the ICE.
- Reconnect the USB cable.

Then please check if the USB option is enabled.

LPT Port Address: The system will search automatically the address of the print port, which is already connected with the hardware. After the connection is successful, WicePlus will also diagnose the hardware right away. (Default address is 378H.)

I/O Wait Times: It depicts the I/O response speed. Increase the value for slower speed and decrease for faster speed. Usually, the value is bigger and the stability is better.

Check ICE Memory: You may also enable this check box to check the condition of the ICE memory.

Long Delay Time: When you are unable to connect your computer to ICE, enable the Long Delay Time check box (see figure above). This will allow a longer handshaking time between your computer and ICE.

Click **OK** button when done.

Reconnection: If a new hardware replaces the current one, it is necessary to reconnect it with PC. Please choose “Connect” from the “Option” main menu for reconnection. Fig. 3-1 would be popped out again. It takes longer waiting time to communicate between PC and the hardware, if the reconnection is done again and again under the same hardware environment.

Note: With the Long Delay Time option enabled, WicePlus handshaking mode with ICE reverts to previous version of WicePlus (i.e., v2.0 or versions released before July 16, 2007).

3.2.2 Code Option Dialog

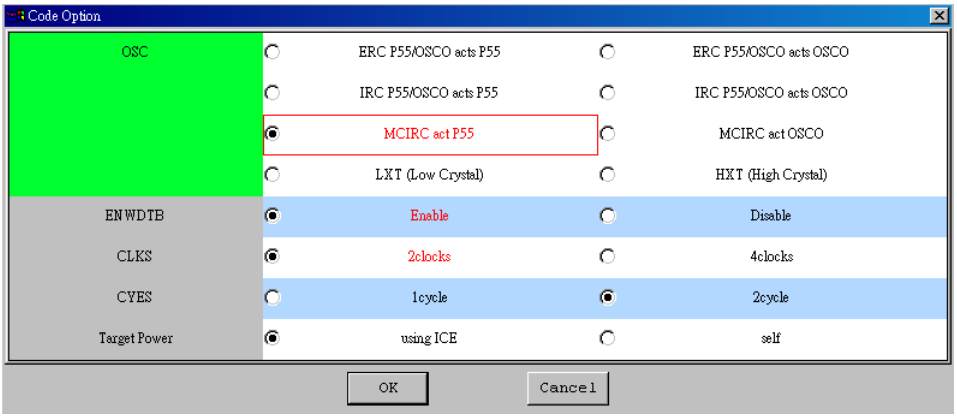


*Fig. 3-2 WicePlus Program **Code** Dialog*

The **Code Option** dialog is displayed next. Check all items to confirm the actual status of the ICE and make appropriate changes as required. Then click **OK** button. Users have to study MCU's or ICE spec to select code option correctly otherwise it can't connect with ICE. Users can enlarge or reduce the dialog's size by drag the edge. WicePlus uses new code option dialog not only USB mode but printer port mode also in version 2.6 or later.

Note: Not all of MCU have to set connect code option

3.2.3 Accelerate Reading Registers Dialog

If you use USB ICE, this dialog will pop up after setting code option. Please refer to 2.3.7 b.

3.3 Create a New Project

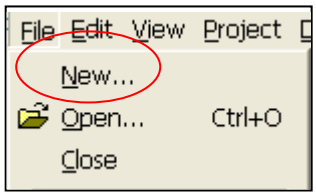
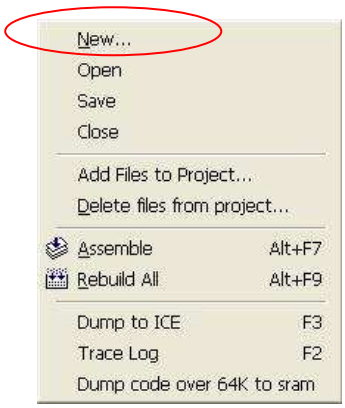


Fig. 3-3 "File" Menu

To create a new project, you need to configure your project with the following steps**:**

1. From the Menu bar click on **File** or **Project** menu and choose **New** command from the resulting pull-down menu.
2. The **New** dialog (shown below) will then display if you have clicked the **New** command from the **File** menu. Otherwise, **New Project** dialog will display (Fig. 3-5) if the **New** command is derived from **Project** menu.



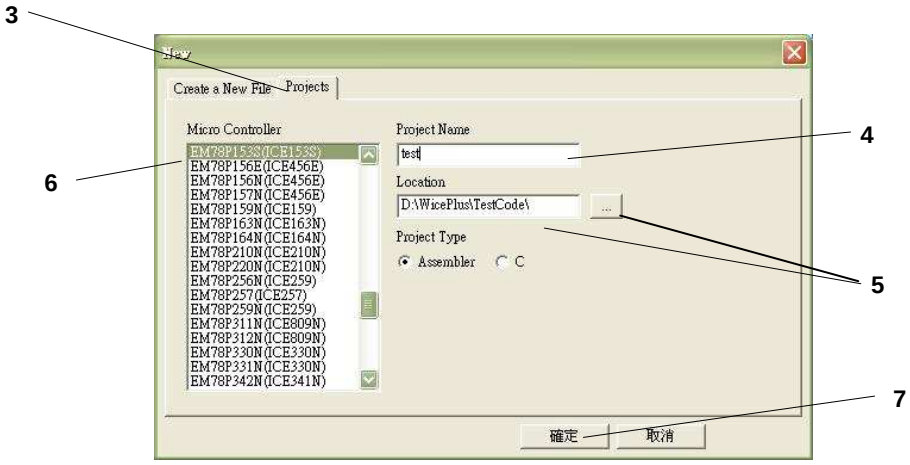


Fig. 3-4 "Project" Menu

Fig. 3-5 "New" Dialog Showing Project Tab for Creating New Project (Derived from **File** Menu)

3. Select **Projects** tab from the **NEW** dialog
4. Assign a name for the new project in the **Project Name** box (suffix ***.prj*** will auto-append to the filename).
5. Locate the folder where you want to store the new project. You may use the **Browse** icon to find the appropriate folder.
6. Select the target microcontroller for your project from the **Micro Controller** list box.
7. Click **OK** button after confirming all your choices and inputs.

The new project is created with the defined project name and microcontroller you have selected is displayed on top of the **Project** window.

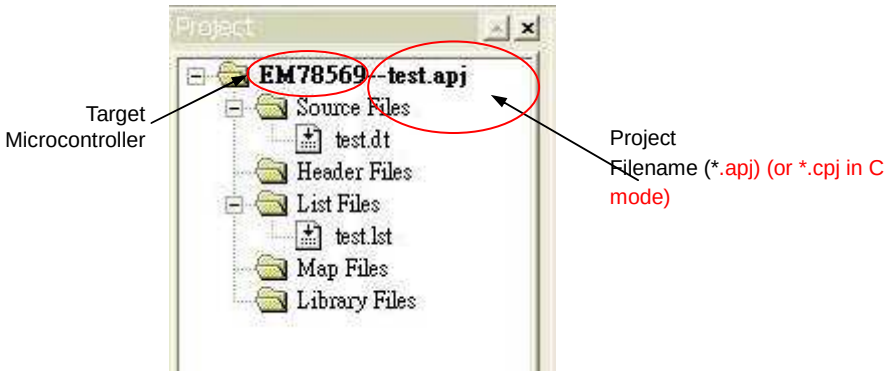


Fig. 3-6 "Project" Window

3.4 Add and Remove Source Files from/to Project

You can either insert existing source files into the new or existing project, or create new ones with WicePlus text Editor and insert them into the project.

3.4.1 Create and Add a New Source File for the Project

If your source file is yet to be created, you can take advantage of the **New** dialog (by clicking **New** command from the File menu) to create your new source file and use the WicePlus text editor to compose its content.

1. Click the **File** tab of the **NEW** dialog and select the type of source file you want to create from the **EMC Source File** list box, i.e., ****.dt*** (default or *.c for C mode) for assembly file; ****.h*** for header file.

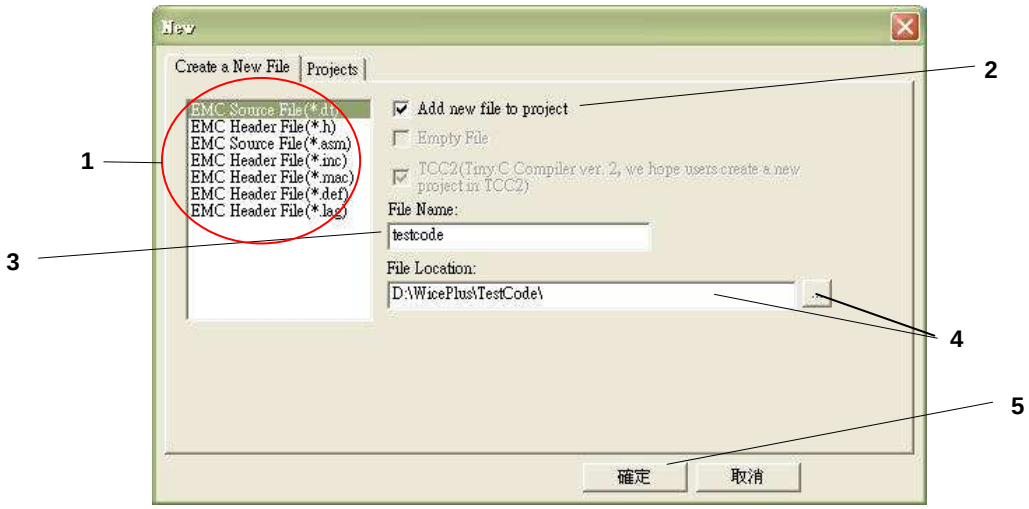


Fig. 3-7 "New" Dialog Showing Project Tab for Creating a New Source File

2. Check **Add to Project** check box (default) if you want to automatically add the new file into your project. Otherwise clear the check box.
3. Assign a filename for the new source file in the **File Name** box.
4. Locate the folder where you want to store the new source file in your disk. You may use the **Browse** icon to find the appropriate folder.
5. Click OK button after confirming your inputs. You will be prompted to start writing the newly defined source file in the **Editor** window.

Note:

1.In C mode, we recommend users to check " Add new file to project" to add the first c file in a new project. WicePlus supplies main() function and interrupt save procedure and interrupt service routine frame and some very important save and restore code in the file for users do develop project and write code

about interrupt easily. We describe the detail about interrupt in sec. 7.10, TCC1 and 7.11, TCC2. Interrupt program is very easy to develop in TCC2. We suggest users use TCC2 instead of TCC1.

```
01 void main()
02 {
03   _asm{MOV A,@0x10
04        CONTW
05   }
06 }
07
08 void _intcall interrupt(void) @ int
09 {
10
11  // Write your code (inline assembly or C) here
12
13 }
14
15 void _intcall interrupt_1(void) @ 0x08:low_int 0
16 {
17   _asm{PAGE @0x0}
18 }
```

Please remember that C compiler just accept one main() function in a project. If users want to add other new files after the first main file was add, users can check “Empty File” to add a empty file.

2. Please don’t write code over column 512, especially in C mode, or serious error will happen. The limit in previous version is about column 256.

3.4.2 Add Existing Source Files to the New Project

If your source file is ready, you can immediately insert it into your new project.

1. From the Menu bar, click on **Project** menu. Choose **Add Files to Project** command from the resulting pull-down menu, and then the **Open** dialog is displayed.

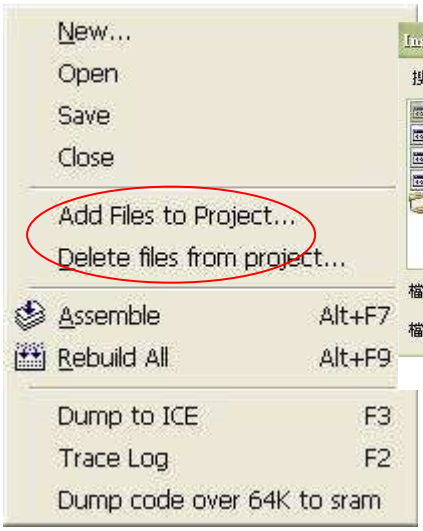


Fig. 3-8a “Add Files to Project” Command

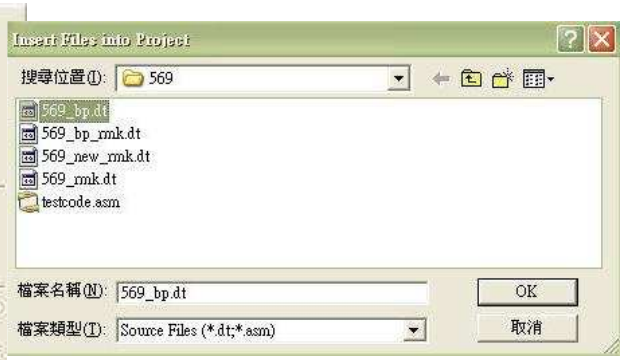


Fig. 3-8b “Open” Dialog

2. Browse and select the file (or multiple files) you intend to insert into the new project.

3. Click **OK** button after confirming your choice.

3.4.3 Deleting Source Files from Project

Fig. 3-9a Deleting Project Files Directly from "Project" Window

1. From the **Project** window, select the file(s) you wish to delete. Then press the **Delete** key from your keyboard.

2. You may also click on the **Delete Files from Project…** command from the **Project** pull-down menu to delete files from project.

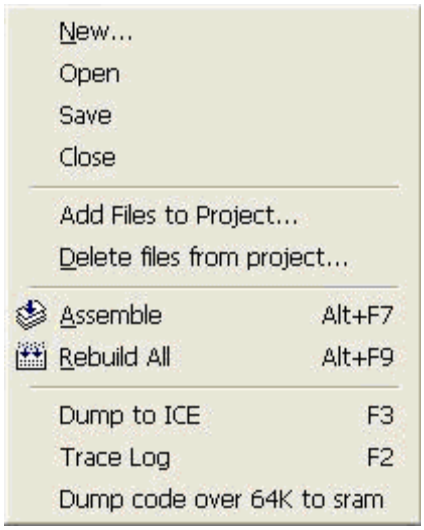


Fig. 3-9b Deleting Project Files from "Project"menu.

3.5 Editing Source Files from Folder/Project

3.5.1 Open Source File from Folder for Editing

You can also open an existing source file in the **Editor** window for a last minute editing before adding it into the new project. To do this–

1. From the Menu bar click on **File** or **Project** menu, choose **Open** command from the resulting pull-down menu.
2. From the resulting **Open** dialog (Fig. 3.8b above) click on the source file and the file is automatically opened in the **Editor** window.

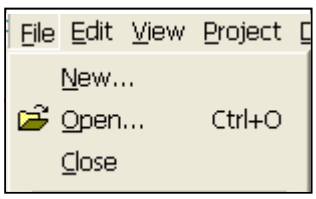


Fig. 3-10 Open &Edit Source File from "File" Menu

To edit source files that are already added into the Project, see next Section.

3.5.2 Open Source File from Project for Editing

You can edit source files that are already inserted in the project. To do so, double click the source file you wish to edit from the **Project** window and the file will open in the **Editor** window.

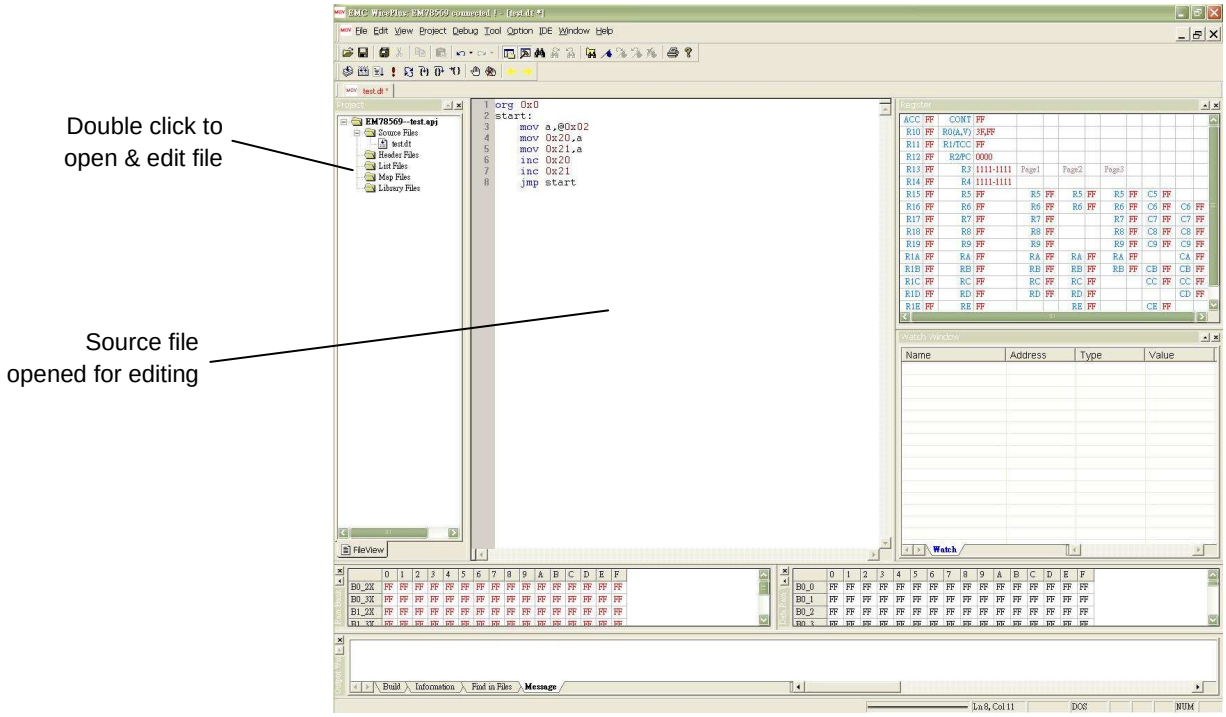


Fig. 3-11 Editing Source File Directly from "Project" window.

3.6 Compile the Project

With your source file(s) embedded into the project, you are now ready to compile your project using the following commands from **Project** menu.

- Click **Assemble (or Compile)** command to compile the active file only (generates **.lst*).
- Click **Rebuild All** command to compile all files in the project regardless of whether they were modified or not.

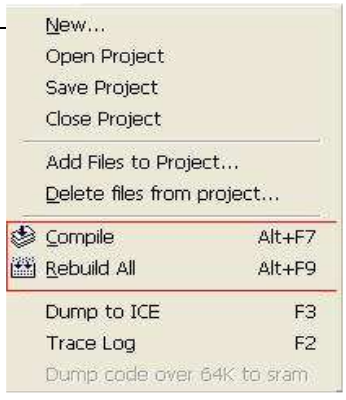


Fig. 3-12 Assemble & Link

Rebuild All will generate objective (**.bbj*) file, list (**.lst*) file, binary (**.cds*) file.

The compiled files are automatically saved in the same folder where your other source files are located. Status of the assembly operation can be monitored from the **Output** window as shown below.

In C mode, rebuild All will generate objective (*.o) file, assembly (*.s) file and binary (*.cds) fils.

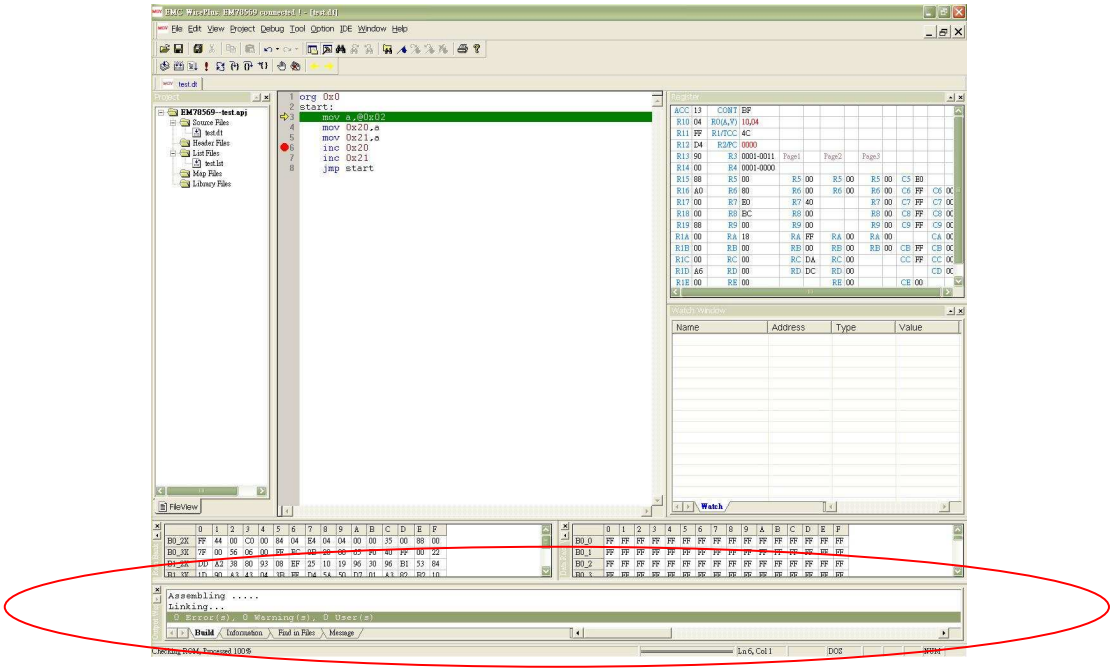


Fig. 3-13 Output Window Showing Successful Compilation

If error is detected during compilation, pertinent error message will also display in the **Output** window with **Build** tag. Double click on the error message to link to the source of error (text line) in the corresponding source file displayed in the **Editor** window. If the corresponding source file is not currently opened, it will open automatically.

Double click to link to the source of error

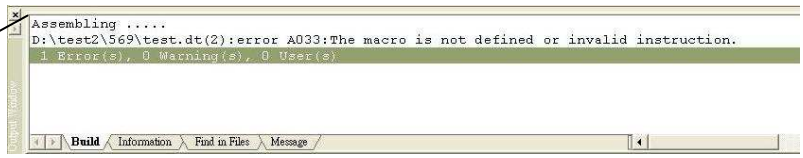


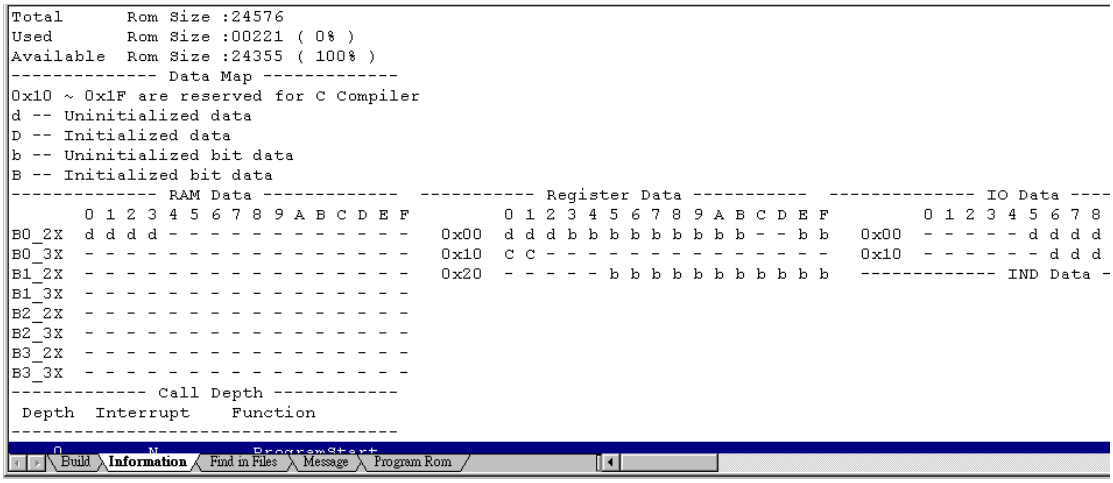
Fig. 3-14 Output Window Showing Compilation Errors

Modify source files to correct the errors and repeat assembling and linking operations.

In C mode, there are many useful messages reported in Information sheet, Output Window if compiling succeeds. For example, it tells users used rom size, available rom size, used ram data in figure, used data register in figure, IO control data in figure, call depth and max call depth. The most important message is the char “C” in register data located 0x10~0x1F. These C chars tell

programmer which and how many common registers to save and restore in interrupt service routine. Please reference to sec 7.5 for the detail.

Fig. 3-14a shows these messages. So we can know that we have to save common register 0x10 and 0x11 when MCU just runs into interrupt service for C system and restore these two common registers before leaving interrupt service.



3-14a Messages after compiling successful

3.7 Dumping the Compiled Program to ICE

With the source files deprived of its errors and successfully compiled, download your compiled program to ICE using the **Dump to ICE** command from **Project** drop-down menu or its corresponding shortcut key (F3).

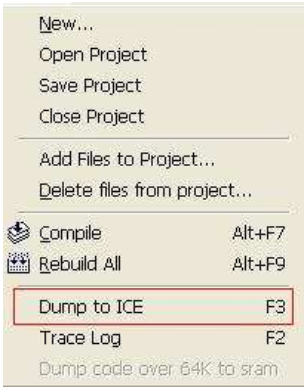


Fig. 3-15 "Dump to ICE" Command

3.8 Debugging a Project

With the compiled program successfully downloaded to ICE, you are now ready to debug the files. Be sure the ICE is properly connected to your computer.

Full debugging commands are available from the **Debug** Menu (shown with its corresponding

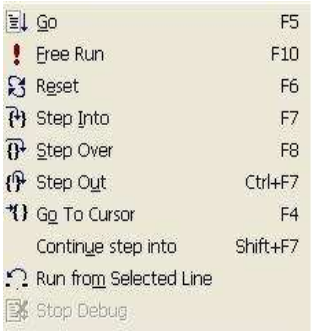


Fig. 3-16b Debugging Commands Drop-Down Menu

shortcut keys in the drop-down menu at right). A number of the frequently used debugging icons are also available from the WicePlus Program Toolbar.

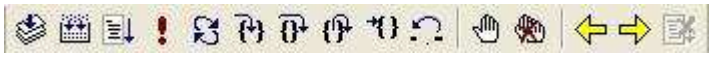


Fig. 3-16a Toolbar for Debugging Commands

F5

Go – Run program starting from the current program counter until breakpoint is matched and breakpoint address is executed.

F10

Free Run – Run program starting from the current program counter until the **"Stop"** button of the "Stop Running" dialog is clicked. All defined breakpoints are ignored while the program is running.

F6

Reset – Perform hardware reset (register contents are displayed with initial values). ICE will return to its initial condition.

F7

Step Into – Execute instructions step-by-step (with register contents updated at the same time).

If RAM of your computer is not enough, don't push it continuously, please.

F8

Step Over – Same as "Step Into" command (see above), but the CALL instruction is executed as "Go" command.

If RAM of your computer is not enough, don't push it continuously, please.

Ctrl + F7

Step Out –Run program starting from the current program counter until the RET / RETI / RETL instruction address is executed.

F4

Go to cursor– To execute from current program counter to the location where the cursor is anchored.

Run from Selected Line – To apply this function, the start for next time will be located on caret.

F9

Toggle Breakpoint – Click with cursor positioned on the line where a breakpoint is going to be set or removed.

Clear All Breakpoints – Remove all already set breakpoints.

Trace Back: After trace log is executed, "trace back" will be acted. The caret moves backward step by step from the last executed address to the address located before the current executed address.

Trace Forth: After trace log is executed, "trace forth" will be acted. The caret moves forward step by step from the last executed address to the address which "trace back" ever went through.

Esc

Stop Debug: Stop Free Run or Go without break point.

During debugging, the contents of Program Counter, Registers, and RAMs are read and displayed each time the program is stopped to provide important interim information during program debugging.

3.8.1 Breakpoints Setting

To assign a breakpoint, position cursor on the line where a breakpoint is going to be set, then double click. Observe the line highlighted in brown.

You can also click on the **Insert/ Remove Breakpoint** icon (hand shape) on the toolbar to set a breakpoint.

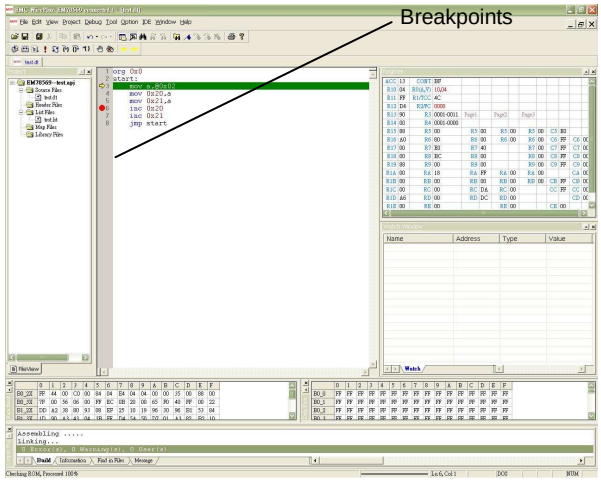
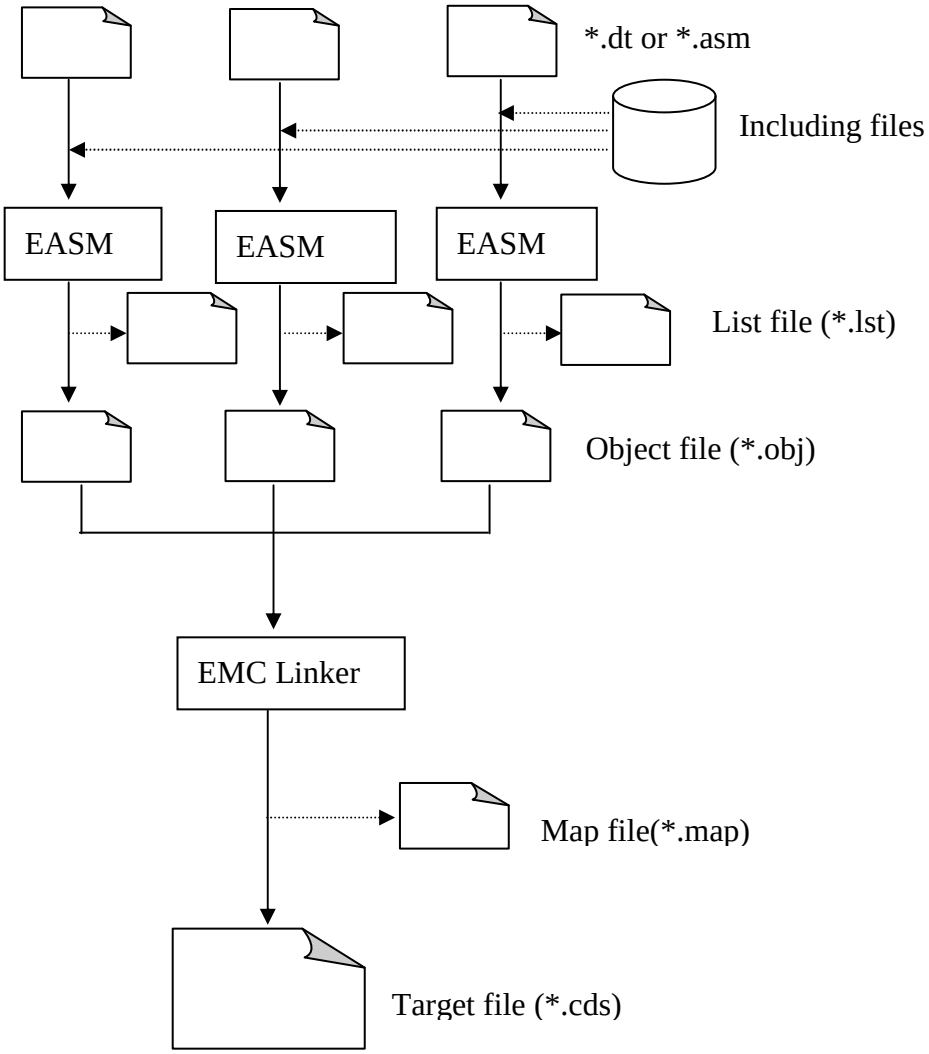


Fig. 3-17 Active Source File with a Defined Breakpoint

Likewise, the defined breakpoint is cleared if you double click on it again, or the hand icon is clicked the second time while the cursor positioned on the defined breakpoint. To clear all existing breakpoints, click **Clear All Breakpoints** command from **Debug** menu.

Chapter 4
Assemble and link

4.1 Assembler and linker process flow



4.2 Statement syntax

[label [:]] operation [operand] [,operand][; comment]

All fields are not case-sensitive, and separated by space or tab.

Label—The [:](colon) is optional and is followed by one or more spaces or tabs.

A label consists of the following characters

A~Z a~z 0~9 _

but with some restrictions :

. 0~9 must not be the first character of a name

. Only the first 31 characters are recognized

. To reserve colon (:) is recommend to programmers, because it could be more readable.

How to define label:

a. zeroflag equ RXX(.YY)

Ex1: zeroflag == R3.2

Ex2: status == R3

b. zeroflag_1 equ 0xXX(.YY)

Ex1: zeroflag_1 == 0x3.2

Ex2: status_1 == 0x3

c. zeroflag_2 equ zeroflag(.ZZ) (zeroflag_1 equ RXX(.YY))

Ex: status_2 == R3

zeroflag_2 == Status_2.2

d. zeroflag_3 equ zeroflag1(.ZZ) (zeroflag equ 0xXX(.YY))

Ex: status_3 == 0x3

zeroflag_3 == Status_3.2

e. Add label with rpage / rbank / iopage

"range" and select "Special Register(R0..R1F)" from Watch Dialog are the same.

"rbank" and select "RAM(bank)" from Watch Dialog are the same.

"iopage" and select "Control Register" from Watch Dialog are the same.

Ex: status_3 == 0x3:rpage 0

zeroflag_3 == status_3.2 (status_3 is already defined in rpage 0)

temp == 0x20:rbank 1

output == 0x6:iopage 1

outputbit_2 == 0x6.2 (output is already defined in iopage1)

***Attention**: 0x3:rpage 0 -> Please don't type space around colon(:).*

f. To get Register or Bit information from label__R(Duplicate _) or label__B:

Ex: zeroflag == 0x3.2

mov a, Zeroflag__R ;equal mov a, 0x3

mov a, zeroflag__B ;equal mov a, 0x2

***Attention**: This method only display to Watch Window automatic. When use this variable, must use register which defined by ICE specification to change register or bank page.*

Operation – An assembler instruction or directive.

Directives give the direction to the assembler.

Examples of instructions:

Example 1: MOV A,@0X20
Example 2: ADD A,@0X20
Example 3:
zeroflag == R3.2
status == R3
carryflag == status.0

```
    org 0x0
    jmp start
start:
    BC zeroflag
    BS status,2
    BC carryflag
    BS status,0
    BC 0x3,2
    BS R3,2
```

Examples of directives:
Example1: ORG 0X20
Example2: END

Operands— One or more operands, separated by commas.
Comment— Comments include line comment and block comment.
Line comment syntax: preceded by a ";" (semi-colon)

Example: MOV A,@0X20 ; move constant value 32 to accumulator

Block comment: /* comment statements */

4.3 Number type

Type	Expression 1	Expression 2	Expression 3
Decimal	**0D<digits>**	**<digits>D**	**<digits>**
Hexadecimal	**0X<digits>**	**<digits>H**	
Octal	**0Q<digits>**	**<digits>Q**	
Binary	**0B<digits>**	**<digits>B**	

P.S. If first digit is A~F or a~f on hexadecimal expression 2, then users must add "0" to precedence of digits.

4.4 Assembler arithmetic operations

The arithmetic result must be calculated after assembler. If the arithmetic expression can not be calculated on assembler time, then display error message. Meanwhile, the arithmetic

expression can not support floating point, so floating point number is self-transferred to integer.

TRUE and FALSE：TRUE is 0xFF，FALSE is 0x00.

The following operators give a summary of the operators, in order of priority.

a. Parentheses (and)

b. Unary operators

! Logical NOT
~ Complement
－Unary minus

c. Multiplication, division, modulo, shift

* Multiplication
/ Division
%Modulo
<<Logical shift left
>>Logical shift right

d. Addition arithmetic operators

＋Addition
－Subtraction

e. Bit AND operator

& Bit AND

f. Bit OR and XOR operators

| Bit OR
^ Bit XOR

g. Logical AND

&& Logical AND

h. Logical OR

|| Logical OR

i. Comparison

== equal
!= not equal
> greater than
< less than
>= greater than or equal to
<= less than or equal to

4.5 Program directives

a. ORG : set value of program counter.
Syntax: **ORG <expression>**

Example: org 0x200

b. EQU or == (Duplicate =): definition constant value
Syntax: **<label> EQU <expression>**

Example 1: R20 equ 0x20
Example 2: R20 == 0x20

c. Comment
Line comment
Syntax: **;** < **string** >

Example: ; this is the comment string.

Block comment
Syntax: **/* <strings> */**

Example: /* this is block comment
example including multi lines */

d. EOP : end of the program ROM page which the EOP instruction belongs to**.**
Syntax: **EOP**

Example:

```
        org   0x10
        mov   0x20,A
        inc   0x20
        eop
        inc   0x20
```

Result after assembling (the first column is address):

```
        org   0x10
0010    mov   0x20,A
0011    inc   0x20
        eop
0400    inc   0x20
```

e. END: the end of program. The rest of program code after END instruction won't be assembled.
Syntax: **END**

Example:

```
        org   0x10
        mov   0x20,a
        inc   0x20
        end
        mov   0x20,a
```

Result after assembling (the first column is address):

```
        org   0x10
0010    mov   0x20,a
0011    inc   0x20
        end
        mov   0x20,a
```

f. PROC, ENDP : definition of subroutine. The directives make a program more readable.

Syntax: **<label> PROC**
<statements>
ENDP

Example:

```
BANK0:   PROC
BC   0X04,6
BC   0X04,7
RET
ENDP
```

P.S. PROC and ENDP directives are only more readable of program, So RET instruction must be existed in subroutine.

g. INCLUDE: Including another source files. The instruction makes the program refined and readable.

INCLUDE function can include system default files and user's defined files.

(a) Include system default files, example for EMC456.INC，EMC32.INC

Syntax: **INCLUDE <filename>**

Example: INCLUDE <EMC456.INC>

(b) Include user's defined files.

Syntax: **INCLUDE "file path + file name"**

Example: INCLUDE "C:\EMC\TEST\TEST456.INC"

The user's defined file must include full path and name.

P.S. Usually including file includes variable defined, macro defined and subroutine defined.

h. PUBLIC and EXTERN: Defined scope of the global label is public or external. Although the WICE software is project oriented, a project can contain two or more files. If the global label is referenced by another file, the global label must be defined to PUBLIC in the defined file, and must be defined to EXTERN in the referenced file.

PUBLIC:

Syntax: **PUBLIC <label>[,<label>]**

EXTERN:

Syntax: **EXTERN <label>[,<label>]**

PUBLIC and EXTERN instructions can be defined in any place of a file which contains one or more PUBLIC or EXTERN instructions.

Example: A project contains two files, one is TEST1.DT; the other is TEST2.DT.
TEST1.DT:

```
        org  0x00
        Public start
        Extern  loop1
Start:
        mov   a,@0x02
        mov  0x20,a
        jmp   loop1
```

TEST2.DT:

```
        org  0x100
        Public  loop1
        Extern  start
Loop1:
        inc  0x20
        jmp  start
```

The label of “Start”, which is defined in the “TEST1.DT” file and is referenced by the “TEST2.DT”file, must be announced as “PUBLIC” in the “TEST1.DT” file and “EXTERN” in the “TEST2.DT” file.

The label of “loop1”, which is defined in the “TEST2.DT” file and is referenced by the “TEST1.DT” file, has to be addressed as “EXTERN” in the ‘TEST1.DT” file and “PUBLIC” in the “TEST2.DT” file.

i. VAR: The instruction defines variable name in the assembly time, so the value of variable is only changed in the assemble time.
Syntax: **Label VAR <expression>**

Example:

```
test var  1
        mov    a,@test
        test   var  test+1
        mov  a,@test
```

j. MACRO, ENDM: The instruction defines macro.
Syntax: **<label> MACRO <parameters>**

```
        statements
ENDM
```

Example:

```
bank0   macro
        bc0x04,6
```

bc0x04,7
endm

***Attention**: The max number of macro is 32*

***Attention**: Those parameters must to be a definite address.*

```
aa_label macro num
    if num >= 0x400
        bc 0x3,6
        bs 0x3,5
    else
        bc 0x3,6
        bc 0x3,5
    endif
endm

org 0xfff
jmp start
org 0x0
start:
    aa_label bb_label
    call bb_label
    aa_label $
    jmp $

org 0x400
bb_label:
     inc 0x20
     ret
```

Example1:

If omit “org 0x400”, “**bb_label”** is an indefinite address before **aa_label** macro expanded.

It will to show the error message “error A037: The operand value can not be calculated.”

```
FCALL  MACRO  addr
    IF  (addr/0x400) != ($/0x400)
        LCALL  addr
    ELSE
        CALL  addr
    ENDIF
ENDM

FJMP  MACRO  addr
    IF  (addr/0x400) != ($/0x400)
        LJMP  addr
    ELSE
        JMP  addr
    ENDIF
ENDM
```

Example2: (EM78P510N)

FCALL and **FJMP** are reserved words. Those two words judge the program code over page or not.

```
FPAGE MACRO NUMB
    IF  NUMB==0
        BC  0x3.5
        BC  0x3.6
    ELSEIF  NUMB==1
        BS  0x3.5
        BC  0x3.6
    ELSEIF  NUMB==2
        BC  0x3.5
        BS  0x3.6
    ELSEIF  NUMB==3
        BS  0x3.5
        BS  0x3.6
    ELSE
        ERROR
    ENDIF
ENDM

FJMP MACRO ADDRESS
    IF ADDRESS/0X400!=$/0X400
        FPAGE   ADDRESS/0X400
    ENDIF
        JMP ADDRESS
ENDM

FCALL MACRO ADDRESS
    IF ADDRESS/0X400!=$/0X400
        FPAGE   ADDRESS/0X400
        CALL    ADDRESS
        FPAGE   $/0X400
    ELSE
        CALL    ADDRESS
    ENDIF
ENDM
```

Example3: (EM78P447S)
FPAGE, FCALL and **FJMP** are reserved words. Those words judge the program code over page or not.

k. MACEXIT: The instruction is only used in the macro defined instructions. If the MACEXIT instruction is assembled, then the rest instructions of macro are not assembled.

Syntax: **MACEXIT**

Example:
Source:

```
test   var   5
bank0   macro
  bc  0x04,6
   if  test>4
      macexit
  endif
  bc  0x04,7
endm
bank0
```

After assembler…(first column is address)

0000 bc 0x04,6

Because “test” variable is equal to five, so the expression “test>4” is true, and the “macexit” instruction is assembled. Because “macexit” instruction is assembled, the rest instructions of macro, “bc 0x04,7”,are not assembled.

l. **MESSAGE:** Display user defined message in the output window.
Syntax: **MESSAGE “<characters>”**

Example:

```
org  0x00
message “ set bank to 0 !!”
 bc    0x04,6
 bc    0x04,7
```

Display below message in the output window after assembler.
USER MESSAGE: set bank to 0!!

***Attention**: The max number of message is 500.*

m. **$:** Current program counter value
“$” is used as an operand.

Example1:

```
jmp   $
```

“jmp $” means that jump same line as dead loop.

Example2:

```
 bc 0x04,6
jmp   $-1
```

“jmp $-1” means that jump back to “bc 0x04,6”

4.6 Conditional assembly

a. **IF :** If the statement after “IF” expression is true, then the following instructions are assembled until “ELSEIF” or “ELSE” or “ENDIF”.
Syntax: **IF <expression>**
<statements1>
[ELSEIF <expression>
<statements2>]
[ELSE
<statements3>]
ENDIF

Example:

```
org 0x00
bank  macro  num
if num==0
   bc  0x04,6
    bc  0x04,7
elseif num==1
    bs  0x04,6
    bc  0x04,7
elseif num==2
    bc  0x04,6
    bs  0x04,7
elseif num==3
    bs  0x04,6
    bs  0x04,7
else
   message “error : bank num over max number!!!”
endif
endm
```

b. IFE: If the statement after “IFE” expression is false, then the following instructions are assembled until “ELSEIFE” or “ELSE” or “ENDIF”.

Syntax: **IFE <expression>**
<statements1>
[ELSEIFE <expression>
<statements2>]
[ELSE
<statements3>]
ENDIF

c. IFDEF: If the statement after “IFDEF” label is defined, then the following instructions are assembled until “ELSEIFDEF” or “ELSE” or “ENDIF”.

Syntax: **IFDEF <label>**
<statements1>
[ELSEIFDEF <label>
<statements2>]
[ELSE
<statements3>]
ENDIF

Example:

```
org 0x00
   ice456  equ  456
ifdef ice456
   bc  0x04,6
   bc  0x04,7
endif
```

d. IFNDEF: If the statement after “IFNDEF” label is not defined, then the following instructions are assembled until “ELSEIFNDEF” or “ELSE” or “ENDIF”.

Syntax: **IFNDEF <label>**
<statements1>
[ELSEIFNDEF <label>
<statements2>]
[ELSE
<statements3>]
ENDIF

4.7 Reserved word

a. directives, operators

+	-	*	/	==
!=	$	@	#	(
)	!	~	%	<<
>>	&	|	^	&&
||	<	<=	>	>=
DS	ELSE	ELSEIF	ELSEIFDEF	ELSEIFE
ELSEIFNDEF	END	ENDIF	ENDM	ENDMOD
ENDP	EQU	EXTERN	IF	IFE
IFDEF	IFNDEF	INCLUDE	MACRO	MACEXIT
MODULE	NOP	PAGE	ORG	PROC
PUBLIC				

b. instructions mnemonics

ADD	AND	BC	BS	CALL
CLR	COM	COMA	CONTR	CONTW
DAA	DEC	DECA	DISI	DJZ
DJZA	ENI	INC	INCA	INT
IOR	IOW	JBC	JBS	JMP
JZ	JZA	LCALL	LJMP	MOV
NOP	OR	RET	RETI	RETL
RLC	RLCA	RRC	RRCA	SLEP
SUB	SWAP	TBL	WDTC	XOR
LCALL	LPAGE	PAGE	BANK	

Note:
Some MCU don’t have LCALL , LJMP, PAGE and BANK instructions, users have to study the specific MCU spec;

c.Register names

A

Chapter 5
Debugger: a source level debugger

The following two steps show how to run the debug environment:

Step 1: Set breakpoint (In the Debug menu, only the "Go" function can enable the breakpoint function, and the other items would disable it automatically during being executed.)

Step 2: The available function of the Debugger menu includes "Go", "Free Run", "Reset", "Step Into", "Step Over" and "Go to Cursor."

5.1 Employ breakpoints

a. What is breakpoint?

Breakpoints are the combinations of addresses and pass counts. A program should be stopped as one of the defined addresses matches with the contents of the program counter and its relative pass count decreases to zero.

b. How to set quick breakpoints?

Step1: Edit a general pass count.

From the Option main menu, choose "Debug option setting". The dialog box of setting debug option appears. Users can type a number from 1 to 255 as a pass count. The default value is 1.

Step2: Set breakpoints.

The first way is to move mouse cursor to a line where a breakpoint is going to be set and then double click by the right button. To set breakpoint is completed as the background color of the clicked line turns into brown. The other way is to move caret to a line, and press the "toggle breakpoint" icon (hand shape) on the toolbar, or choose "Toggle Breakpoint" from the Debug menu.

c. How to set user-defined breakpoints.

Step1: Set the contents of the pass count to 0.

From the Option main menu, choose "Debug option setting". The dialog box of setting debug option appears. Users must reset the pass count to zero.

Step2: Set breakpoint.

Users can move mouse cursor to a line where a breakpoint is going to be set and then double click by the right button. User should enter a suitable number for the pass count as the dialog box of set breakpoint counter appears. To set breakpoint is completed as the background color of the clicked line turn to brown.

5.2 To debug a project

The contents of program counter, registers and RAMs are read and displayed each time as program stopped. They provide programmers very important information during debugging program. There are several useful functions as shown below in the Debug menu. The function keys or hot keys are shown in the parentheses.

a. Go (F5): Program runs from current program counter until breakpoint is satisfied, and the breakpoint address is executed.

b. Free Run (F10): Program runs from current program counter until the execution is stopped. Meanwhile all defined breakpoints are ignored during running.

c. Reset (F6): To reset ICE to the initial conditions

d. Step Into (F7): Single step execution.

e. Step Over (F8): Single step execution, and the subroutine skipped.

f. **Step Out (Ctrl+F7):** Program runs from current program counter until exit subroutine.

g. Go To Cursor (F4): Executing from current PC to the location where the cursor is anchored.

Chapter 6
C Fundamental Elements

6.1 Comments

For a single line comment**:**

//	All data in the line after the comment symbol (twin-slash mark) will be ignored.

For Multi line comments**:**

/* … */	All data in the line located within the comment symbols (slash mark + asterisk) will be ignored.

Comments are used to help you understand the program code. It can be placed anywhere in the source program. The compiler will ignore the comment part from the source code, thus no extra memory is required in the program execution.

Example:

```
// This is a single line comment

/*
This is the comment line 1
This is the comment line 2 */
```

6.2 Reserved Words

The reserved words for WicePlus C Compiler are made up of both the ANSI C conformity reserved words and the EM78 Series unique reserved words. The following table summarizes all the applicable reserved words for this compiler.

ANSI C Conformity Words					
const	default	goto	switch	typedef	sizeof
break	do	if	short	union	extern
case	else	int	signed	unsigned	
char	enum	long	static	void	

continue	for	return	struct	while	
EM78 Series Unique Words					
indir	ind	page	on	off	
io	iopage	_intcall	rpage		
low_int	_asm	bit	bank		

NOTE

- ***Double*** *and* ***float*** *are* ***NOT*** *supported by the EM78 Series C Compiler.*
- *_asm is added for the EM78 series C compiler. Don't use _ASM.*
- *indir, ind, io, iopage, rpage are for MCU hardware definition and declaration.*

6.3 Preprocessor Directives

Preprocessor directives always begin with a pound sign (#). The directives are recognized and interpreted by the preprocessor in order to compile the source code properly.

6.3.1 #include

#include "file_name":	The preprocessor will search the working directory to find the file.
#include <file_name>:	The preprocessor will search through the working directory first to look for the file. If the file cannot be found in the working directory, it will search the file from the directory specified by the environment variable EMC_INCLUDE.

#include tells the preprocessor to add the contents of a header file into the source program.

NOTE

- *We don't suggest users to include c file. Maybe compiler will meet errors it users include c file.*
- *Suppose that uaa is declared global, unsigned int (unsigned int uaa) variable in headfile.h. Now uaa is used in testcode.c. If you want to use uaa in kkdr.c, first you have to declare extern unsigned int uaa before you use it in kkdr.c file. The same way to use in the third or more others c source file that are included in the same project.*

Example 1:

```
#include <EM78.h>
#include "project.h"
```

```
#include “ad.c”    // It may meet errors.
```

Example 2:

```
unsigned int uaa;     //in headfile.h
…

#include ”headfile.h”  //in testcode.c file
#include “kkdr.h”      //in testcode.c file
#include “pprr.h”      //in testcode.c file
main ()                //in testcode.c file
{
   uaa=0x21;
   test1();
   test2();
…
}

test1();                  //  in kkdr.h file
..

#include “pprr.h”         //in kkdr.c file
extern unsigned int uaa;  //in kkdr.c file
void test1()              //in kkdr.c file
{
   uaa=0x38;
   test2();
……
   uaa=0x43;

}

test2()                   //in pprr.h file
……

extern unsigned int uaa;  //in pprr.c file
void test2()              //in pprr.c file
{
   uaa=0x29;
}
```

6.3.2 #define

#define identifier
#define identifier token_list
#define identifier (parameter_list) token_list
#define identifier( ) token_list

The #define directive is used to define a string constant which will be substituted into source code by the preprocessor. It makes the source program more legible.

NOTE

Multi-line macro definition should be cascaded with a backslash (\ ) in between the lines. When using assembly code in macro, use ONLY one instruction in a line.

Example:

```
#define MAXVAUE 10
#define sqr2(x, y) x * x + y * y
```

6.3.3 #if, #else, #elif, #endif

#if constant_expression
#else
#elif constant_expression
#endif

The #if directive is used for conditional compilation. It should be terminated by #endif. #else can be used to provide an alternative compilation. If necessary, the program can use #elif for an alternative compilation which should only be used for valid expressions.

Example:

```
#define RAM 30
#if (RAM < 10)
        #define MAXVALUE 0
#elif (RAM < 30)
        #define MAXVALUE 10
#else
        #define MAXVALUE 30
#endif
```

6.3.4 #ifdef, #ifndef

#ifdef identifier
#ifndef identifier

The #ifdef directive is used for conditional compilation of definitions for the identifier. The #ifndef directive is used when conditionally compiling codes with the specified symbol not defined. Both these two directives must be terminated by #endif and can be optionally used with #else.

Example:

```
#define DEBUG   1
#ifdef (DEBUG)
        #define MAXVALUE 10
#else
        #define MAXVALUE 1000
#endif
```

6.4 Literal Constants

6.4.1 Numeric Constant

Decimal:	Default
Hexadecimal constant:	Digit prefix with “0x”
Binary digit:	Digit prefix with “0b”

Numeric constants can be presented in decimal and hexadecimal, depending on the prefix modifier. Binary and octonary numerics are not supported.

Example:

```
12, 34      // Decimal
0x5A, 0xB2  // Hexadecimal
0b10111001  // Binary digit
```

6.4.2 Character Constant

‘character’

Character constants are denoted by a single character enclosed by single quotes. ANSI C Escape Sequences as shown below are treated as a single character.

ANSI C Escape Sequence		
Escape Character	**Meaning**	**Hexadecimal**
\0	Null	00
\a	Bell (Alert)	07
\b	Backspace	08
\f	Form Feed	0C
\n	New Line	0A
\r	Carriage Return	0D
\t	Horizontal Tab	09
\v	Vertical Tab	0B
\\	Backslash	5C
\?	Question Mark	3F
\’	Single Quote	27
\”	Double Quote	22

Example:

```
‘a’, ‘b’,’c’, /x00
```

6.4.3 String Constant

“character_list”

String constants are series of characters enclosed in double quotes, and which have an implied null value (‘\0’) after the last character.

NOTE

It takes one more character space for constant string to store the null value.

Example:

```
“Hello World”
“Elan Micro”
```

6.5 Data Type

The size and range (maximum and minimum values) of the basic data type are as shown below.

Type	Range	Storage Size (Byte)
void	N/A	None
(signed) char	–128 ~ 127	1
unsigned char	0 ~ 255	1
(signed) int	-128 ~ 127	1
unsigned int	0 ~ 255	1
(signed) short	–32768 ~ 32767	2
unsigned short	0 ~ 65535	2
(signed) long	–2147483648 ~ 2147483647	4
unsigned long	0 ~ 4294967295	4
bit	0 ~ 1	1 (Bit)

NOTE

1. ***Floating*** *and* ***Double*** *types are not supported.*
2. *See Section 5.4 of Chapter 5for more details on “Bit Data Type”.*
3. *If user use* ***long*** *data type for multiplication,division,modulus,compare operation , 0x20 ~ 0x24 (5 bytes) of bank 0 are occupied by compiler. Therefore , don’t assign these address to any variable when you do those operations.*

When an arithmetic operator, such as, “*”, “/”, and “%” is used with different data types, conversion of right-aligned variables to left-aligned data type is done before the operator takes effect. We suggest users use the same data type to develop program.

Example:

```
Int I1, I2;
Short S1, S2, S3;
Long L1, L2;
I1 = 0x11;
I2 = 0x22;
S1 = I1 * I2;  ➔  change to S1 = (short) I1 * (short) I2;
          // If forgot to add "(short)" before I1 and I2, the final
          // result (in S1) will be 1 byte only
S1 = 0x1111;
S2 = 0x02;
L1 = S1 / S2;  ➔  change to L1 = (long) S1 * (long) S2;
          // If forgot to add "(long)" before S1 and S2, the final
          // result (in L1) will be 2 bytes only.
```

Tiny C compiler uses two's compliment to perform mathematical negative signed declaration.

Example:

```
int abc;
unsigned int uir;
abc=-0x18;
Assume abc is allocated at 0x20, bank 1. Then in Ram bank window
you will see E8 at that location.
```

6.6 Enumeration

enum identifier
enum idenftifier {enumeration-list [=int_value]...}
enum {enumeration-list}

Enumeration defines a series of named integer constants. With the definition, the integer constants are grouped together with a valid name. For each name enumerated, you can specify a distinct value.

Example:

```
enum tagLedGroup {LedOff, LedOn} LEDStatus;
```

6.7 Structure and Union

struct (union)-type-name:
struct (union) identifier
struct (union) identifier {member-declaration-list}
struct (union) member-declaration-list

member-declaration-list:

member-declaration
member-declaration-list member-declaration

member-declaration:
member-declaration-specifiers declaration-list

member-declaration-specifiers:
member-declaration-specifier
member-declaration-specifiers member-declaration-specifier

The structure groups related data and each data in the structure can be accessed through a common name. Unions are groups of variables that share the same memory space.

NOTE

- *Do not use bit data type in structure and union, in stead, use bit field.*
- *Structure and union cannot be used in function parameter.*

Example 1:

```
struct st
{
   unsigned int b0:1;
   unsigned int b1:1;
   unsigned int b2:1;
   unsigned int b3:1;
   unsigned int b4:1;
   unsigned int b5:1;
   unsigned int b6:1;
   unsigned int b7:1;
};
struct st R5@0x05: rpage 0;  //struct R5 is related to 0x05, rpage
0
```

Example 2:

```
struct tagSpeechInfo{
   short rate;
   long size;
} SpeechInfo;
union tagTest{
   char Test[2];
   long RWport;
} Test;
```

6.8 Array

declarator:
array-declarator:

array-declarator
[constant-expression]

array-declarator [constant-expression]

Array is a collection of same type data and can be accessed with the same name.

NOTE

- *If "const" is used to declare an array, the data will be placed at the program ROM.*
- *The maximum size of an array is 32 bytes (RAM bank).*

Example:

```
int array1 [3] [10]
char port [4]
const int myarr [2] = {0x11, 0x22};
            // 0x11, 0x22 will be put at program rom
```

6.9 Pointer

declarator
type-qualifier-list * declarator

A pointer is an index which holds the location of another data or a NULL constant. All types of pointer occupy 1 byte.

NOTE

Function pointer is not supported.

Example:

```
int *pt;
```

6.10 Operators

6.10.1 Types of Supported Operators

The supported operators for the C expression are as follows**:**

- Arithmetic operators
- Increment and decrement operators
- Assignment operators
- Logical operators
- Bitwise operators
- Equality and relational operators
- Compound assignment operators

The table below shows the detailed description of each of the operators**:**

Arithmetic Operators		
Symbol	**Function**	**Expression**
+	addition	expr1 + expr2

–	subtraction	expr1 – expr2
*	multiplication	expr1 * expr2
/	division	expr1 / expr2
%	modulo	expr1 % expr2

Increment Operators		
Symbol	**Function**	**Expression**
++	increase by 1	expr ++
--	decrease by 1	expr --

Assignment Operators		
Symbol	**Function**	**Expression**
=	equal	expr1 = expr2

Bitwise Operators		
Symbol	**Function**	**Expression**
&	bitwise AND	expr1 & epxr2
|	bitwise OR	expr1 | expr2
~	bitwise NOT	~expr
>>	right shift	expr1 >> expr2
<<	left shift	expr1 << expr2
^	bitwise XOR	expr1^expr2

Equality, Relational, and Logical Operators			
Symbol	**Function**	**Expression**	**Example**
<	Less than	expr < expr	x < y
<=	Less than or equal	expr <= expr	x <= y
>	Greater than	expr > expr	x > y
>=	Greater than or equal	expr >= expr	x >= y
==	Equality	expr == expr	x == y
!=	Inequality	expr != expr	x != y
&&	Logic AND	expr && expr	x && y
||	Logic OR	expr || expr	x || y
!	Logic NOT	!expr	!x

Compound Assignment Operators		
Symbol	**Function**	**Example**
+=	y=y + x	x += y
-=	y = y - x	x -= y
<<=	y = y << x	y<<=x
>>=	y= y >> x	y>>=x
&=	y= y & x	y&=x
^=	y= y ^ x	y^=x
|=	y= y | x	y |= x

6.10.2 Prefix of Operators

Priority	Same Level Operators, from Left To Right
Highest	( ) [ ] - >
↕	! ~ ++ -- -(unary) +(unary) (type_cast) *(indirection) & (address) sizeof
↕	* / %
↕	+ -
↕	<< >>
↕	< <= > >=
↕	== !=
↕	&
↕	^
↕	|
↕	&&
↕	||
↕	?:
↕	= += -= *= /= %= >>= <<= &= |= ^=
Lowest	,

6.11 If-else Statement

```
if (expression) statement
else statement
```

"If" statement executes the block of codes associated with it when the evaluated condition is true. It is optional to have an "else" block which will be executed when the evaluated condition is false.

Example:

```
if (flag == 1)
{
      timeout=1;
      flag=0;
}
else timeout=0;
```

6.12 Switch Statement

```
switch (expression)
{
        case const-expr: statements
        case const-expr: statements
        default: statements
}
```

"Switch" statement is flexible to be set with multiple branches depending on a single evaluated value.

> **NOTE**
> *The expression will be checked as INT type, thus only 256 cases can be used in a switch.*

Example:

```
switch (I)
{
        case 0: function0();
          break;
        case 1: function1();
          break;
        case 2: function2();
          break;
        default: funerror();
}
```

6.13 While Statement

while (expression) statement

"While" statement will check the expression first, if the expression is true it will then execute the statement.

Example:

```
while (value != 0)
{
      value--;
      count++;
}
```

6.14 Do-while Statement

do
{
statement
} while (expression);

"Do-while" will first execute the statement and then check the expression. If the expression remains true, then it proceeds to the statement until the expression becomes FALSE.

Example:

```
do {
         value --;
         count++;
} while (value != 0);
```

6.15 For Statement

for (expr1; expr2; expr3) statement;

"For" statement is equivalent to the following statement**:**

```
expr1;
while (expr2)
{
        statement;
        expr3;
}
```

"expr1" is executed first. Normally "expr1" will be the initial condition. "While" statement is executed in the same manner.

Example:

```
for (i = 0; i < 10; i++)
{
         value = value + i;
}
```

6.16 Break and Continue Statements

break;
continue;

The "break" statement exits from the innermost loop or switch block. The "continue" statement on the other hand will skip the remaining part of the loop and jump to the next iteration of the loop. "Continue" is useful in loop statements but it cannot be used in switch loops.

Example:

```
break exampl see switch.

for (i = 0; i < 10; i++)
{
         flag = indata(port);
         if (flag == 0) continue;
         outdata(port);
}
```

6.17 Goto Statement

```
goto label;
    …
label: …
```

"goto" statement is used to jump to any place of a function. It is useful to skip from a deep loop.

Example:

```
for (i = 0; i < 10; i++)
        for (j = 0; j < 100; j++)
                for (k = 0; k < 100; k++)
                {
                flag = crccheck(buffer);
                if (flag != 0) goto error;
                outbuf(buffer);
                }
error:
        //clear up buffer;
```

6.18 Function

Function is the basic block of the C language. It includes function prototype and function definition.

6.18.1 Function Prototype

```
<return_type> < function_name>  (<parameter_list>);
```

A "function prototype" should be declared before the function can be called. It contains the return value, function name, and parameter types.

NOTE

- *The total parameters passed to a function should be a fixed number. The compiler does not support uncertain parameter_list.*
- *Recursive functions are not supported in the compiler.*
- *Do not use "struct" or "union" as the parameter for function.*
- *Function pointer is not supported.*
- *Bit data type cannot be used as a return value.*
- *For reduced ram bank wastage ,We suggest users using global variable in function instead of using argument.*

Example (Function Prototype):

```
unsigned char sum(unsigned char a,unsigned char b);
…
```

6.18.2 Function Definition

```
<return_type> < function_name>  (<parameter_list>)
{
     statements
}
```

Example (Function Content):

```
unsigned char sum(unsigned char a,unsigned char b)
{
         return (a+b);
}
```

Chapter 7
C Control Hardware Related Programming

7.1 Register Page (rpage)

<variable name> @<address>[: rpage <register page number >];

The data type is used to declare a variable at a certain register page. Users have to declare clearly which register page is, including rpage 0.

NOTE

- *If a variable is declared as "rpage," it cannot be declared as "bank," "iopage," or "indir" at the same time.*
- *Only global variable can be declared as "rpage" data type.*
- *Although an MCU just has rpage 0 , but <register page number> must be assigned.*
- *Some MCU use register bank instead of register page in special register, the same, users have to declare in rpage .*
- *Usually common registers 0x10~0x1F are reserve for C compiler. But users can declare a variable in 0x10~0x1F with rpage 0 if C Compiler doesn't use. For example, we declare unsigned int uiR16 @ 0x16 : rpage 0 if C compiler doesn't occupy. But if C Compiler uses 0x16, it will report error and tell us how many continuous common registers space are needed.*

Example:

```
unsigned int myReg1 @0x03: rpage 0;
        // myReg1 is at address 0x03 of register page 0
        // Although the specific register only have one register
        //page,the register page number cannot be ignored.

unsigned int myReg2 @0x05: rpage 1;
        // myTest is at address 0x05 of register page 1
        // If the specific register have more than one register
        // page, user should point out in which register page the
        // variable is located.

struct st
{
    unsigned int b0:1;
    unsigned int b1:1;
    unsigned int b2:1;
    unsigned int b3:1;
    unsigned int b4:1;
```

```
    unsigned int b5:1;
    unsigned int b6:1;
    unsigned int b7:1;
};
struct st myReg3@0x06: rpage 0;
```

CONT	
R0(A, V)	
R1/TCC	
R2/PC	
R3	***myReg1***
R4	

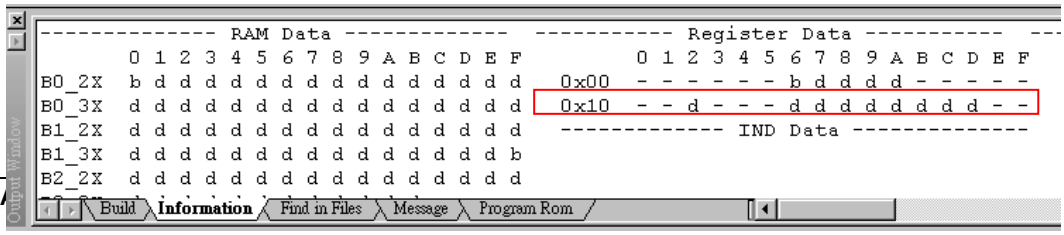
	rpage 0	**rpage 1** ...		**iopage 0**	**ioage 1** ...
R5		***myReg2***	IOC5		
R6	***myReg3***		IOC6		
R7			IOC7		
R8			IOC8		
R9			IOC9		
RA			IOCA		
RB			IOCB		
RC			IOCC		
RD			IOCD		
RE			IOCE		
RF			IOCF		

Declare variables in common registers 0x10~0x1F

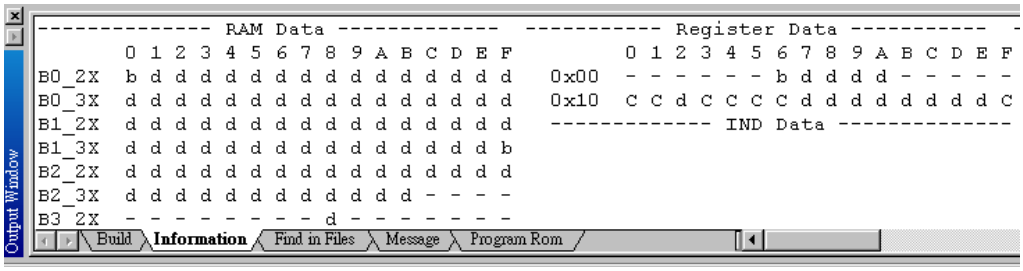
Example:

```
unsigned int uiR12 @ 0x12: rpage 0; //declare uiR12 at common
        register 0x12
unsigned int uiRDD @ 0x16: rpage 0;
unsigned int uiR17 @ 0x17: rpage 0;
unsigned int uiR18 @ 0x18: rpage 0;
unsigned int uiR19 @ 0x19: rpage 0;
unsigned int uiR1A @ 0x1A: rpage 0;
unsigned int uiR1B @ 0x1B: rpage 0;
unsigned int uiR1C @ 0x1C: rpage 0;
unsigned int uiR1D @ 0x1D: rpage 0;

…
Now there is a error message in Build sheet, Output Window,and we
know which un-used common registers (character '-' ) in Information
sheet, Output Window.
Error LNK1116: Can't allocate space for C Register 'ecx' at RPage
        0 : 0x10 - 0x1F, which need 4 byte(s).
```


RAM Data
Register Data
IND Data
0 1 2 3 4 5 6 7 8 9 A B C D E F
B0_2X b d d d d d d d d d d d d d d d
B0_3X d d d d d d d d d d d d d d d d
B1_2X d d d d d d d d d d d d d d d d
B1_3X d d d d d d d d d d d d d d d b
B2_2X d d d d d d d d d d d d d d d d
0x00 - - - - - - b d d d d - - - - -
0x10 - - d - - - d d d d d d d d - -
Output Window
Build
Information
Find in Files
Message
Program Rom

Error message LNK1116 tells us C Compiler need 4 continuous bytes. So we can't declare our variable at 0x12 or 0x16. Now we declare uiRDD at 0x1E instead 0x16.
Unsigned int uiRDD @ 0x1E :rpage 0;
Now no more error message like LNK1116 shows and compiling succeed. Character 'C' means C Compiler uses, 'd' means data type occupied, 'b' means bit data type, '-' means unused.



Assume there is an error message :
Error LNK1116: Can't allocate space for C Register 'pr0' at RPage 0 : 0x10 - 0x1F, which need 2 byte(s).
Now we have to adjust a continuous, 2 bytes space in 0x10~0x1F for it.

7.2 I/O Control Page (iopage)

io <variable name> [@<address>[: iopage <io control page number>]];

Declare the variable at the register page it is located. Users have to declare clearly that the io variable is located at which iopage, though there is only one io control page.

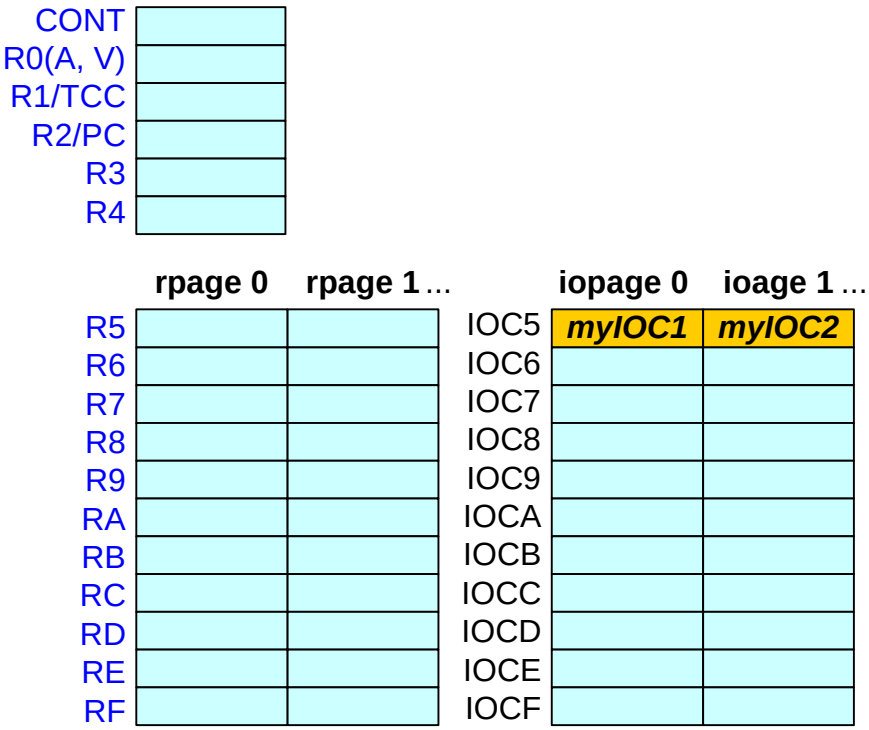
NOTE

- *If a variable is declared as "iopage," it cannot be declared as "bank," "rpage," or "IND" at the same time.*
- *Only global variable can be declared as "iopage" data type.*
- *Although an MCU just has iopage 0 , but <io control page number > must be assigned.*

Example:

```
io unsigned int myIOC1 @0x05: iopage 0;
        // myIOC1 is at address 0x05 of io control page 0

io unsigned int myIOC2 @0x05: iopage 1;
        // myIOC2 is at address 0x05 of io control page 1
```



7.3 Ram Bank

<variable name> [@<address>[: bank <bank number>]];

Declare the variable at which RAM bank it is located. The <bank number > has to be indicated, including variable is declare at at Bank 0.

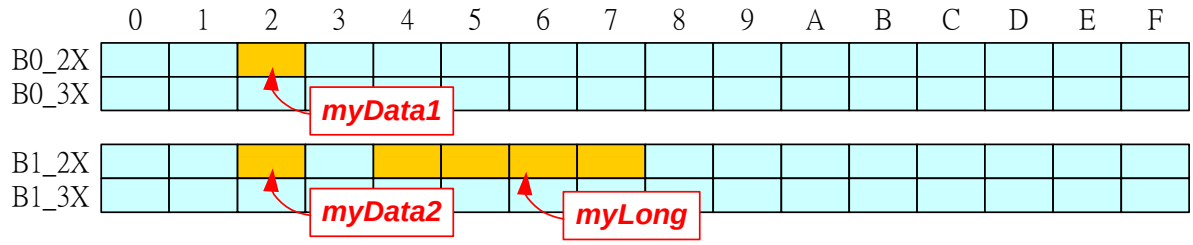
NOTE

- *If a variable is declared as "bank," it cannot be declared as "rpage," "iopage," or "indir" at the same time.*
- *Only global variable can be declared as "bank" data type.*

Example:

```
unsigned int myData1 @0x22: bank 0;
        // myData1 is at address 0x22 of ram bank 0
unsigned int myData2 @0x22: bank 1;
        // myData2 is at address 0x22 of ram bank 1
unsigned short myshort @0x20: bank 0;
         // myshort is at address 0x20 and 0x21 of ram bank 0
unsigned long myLong @0x24: bank 1;
        // myLong is at address 0x24~0x27 of ram bank 1
```

RAM Bank:



7.4 Bit Data Type

bit <variable name> [@<address> [@bitsequence] [: bank <bank number> / rpage <page number>]];

Bit data type occupies only one bit.

NOTE

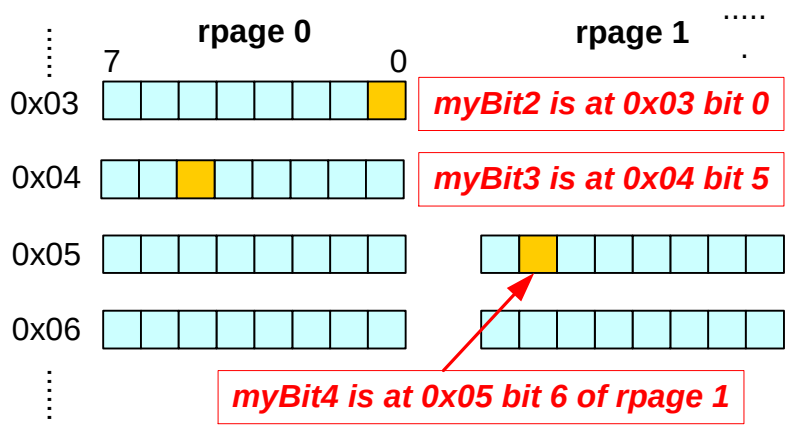
- *Bit data type cannot be used in struct and union. It is recommended to use bitfield in struct and union, such as:*

```
union mybit {
        unsigned int b0:1
        unsigned int b1:1
        unsigned int b2:1
        unsigned int b3:1
        unsigned int b4:1
        unsigned int b5:1
        unsigned int b6:1
        unsigned int b7:1
};
```

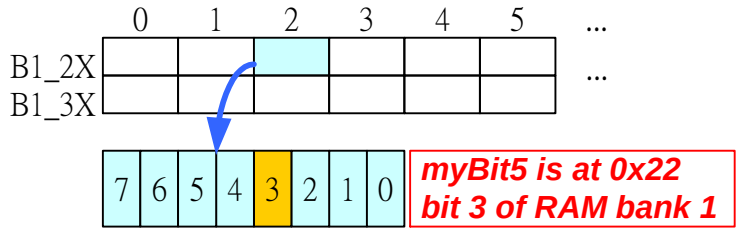
- *Bit data type cannot be used in function parameter.*
- *Bit data type cannot be used as a return value.*
- *Bit data type cannot be operated by arithmetic operator with other data type.*
- *Bit data type is not supported in the IO control register.*
- *Bit is a reserved word, so DON NOT use it as a name of "struct" or "union".*
- *Only global variable can be declared as "bit" data type.*
- *Users can't assign location for Bit data in local field, otherwise compiler will occur error.*

Example:

```
bit myBit1;                          // location of myBit1 is assigned by
                                     // linker
bit myBit2 @0x03 :rpage 0;
                                     // if doesn't declare bit sequence,
                                     // the default location is at bit 0.
                                     // Therefore myBit2 is at bit 0 of
                                     // 0x03 of rpage 0
bit myBit3 @0x04 @5: rpage 1;        // myBit3 is at bit 5 of 0x04, rpage
                                     // 1
bit myBit4 @0x05 @6: rpage 1;        // myBit4 is at 0x05 bit 6 of rpage
                                     // 1
bit myBit5 @0x22 @3: bank 1;         // myBit5 is at 0x22 bit 3 of ram
                                     // bank 1
```



RAM Bank:



7.5 Data/LCD RAM Indirect Addressing

indir <variable name> [@<address>[: ind <ind number>]];

Declare the variable at which indirect data RAM or LCD ram is located. The <ind number > has to be indicated if address is assigned.

If the MCU has Data RAM, use “ind 0” (indirect RAM 0)

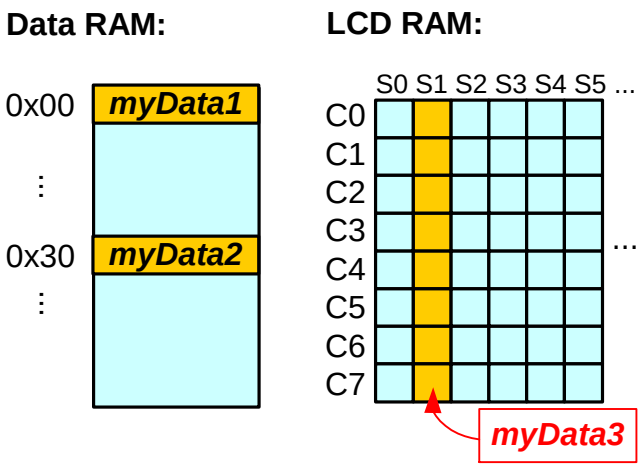
If the MCU has an LCD RAM, use “ind 1” (indirect RAM 1)

NOTE

- *If the specified MCU does not support IND bank, the compiler will generate an error message, e.g., “Symbol ‘WriteIND’ undefined”.*
- *Only global variable can be declared as “indir” data type.*
- *Indir data type does not support array or point variable.*

Example:

```
indir int nData1;
                  //default is “ind 0”, so nData1 is at Data Ram
indir int nData2 @0x30: ind 0;
                  //nData2 is at Data Ram because using “ind 0”.
indir int nData3 @0x01: ind 1;
                  //nData3 is at LCD Ram because using “ind 1”.
```



7.6 Allocating C Function to Program ROM

```
<return value> <function name>(<parameter list>) @<address> [: page <page number>]
{
        ......
}
```

You can place function at the dedicated address of the program ROM, and use “page” instruction to allocate which page in the program ROM you wish to assign.

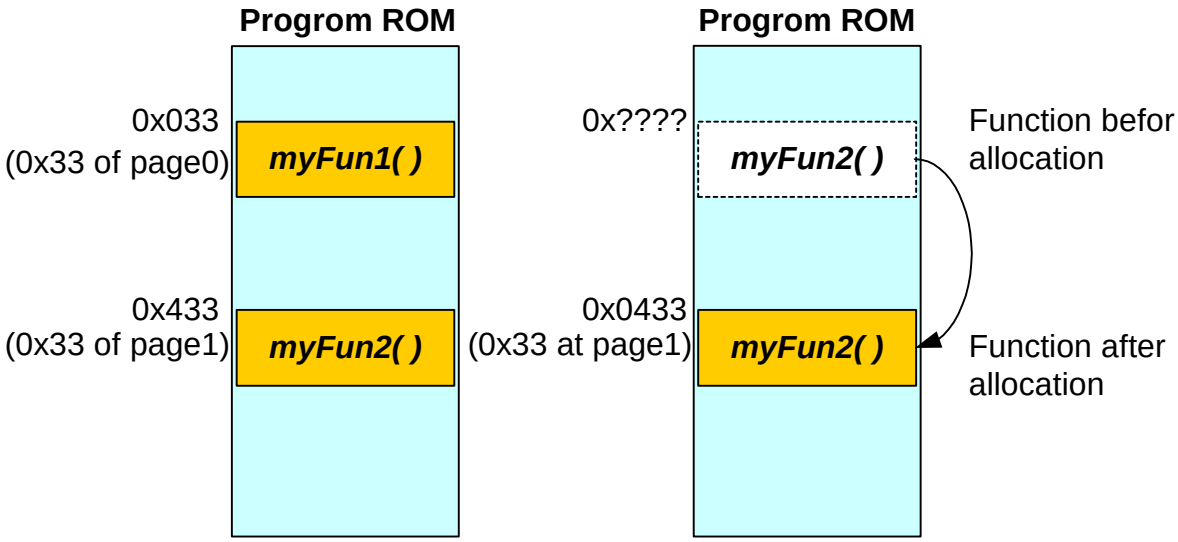
NOTE

- *Only functions can be declared as “page.”*

- *Don not allocate the interrupt save procedure nor interrupt service routine at the dedicated address of the program ROM.*

Example:

```
void myFun1(int x, int y) @0x33
          // myFun1() is put at 0x33 at ROM page 0 (default page)
{
          ……
}
void myFun2(int x, int y) @0x33: page 1
          //myFun2() is put at 0x33 at ROM page 1
{
          ……
}
```



7.7 Putting Data in ROM

const <variable name>;

Some data cannot be altered during program execution. Hence, you need to store such data into the program ROM to save limited RAM space. The Compiler uses the "TBL" instruction to incorporate such data into the program ROM.

NOTE

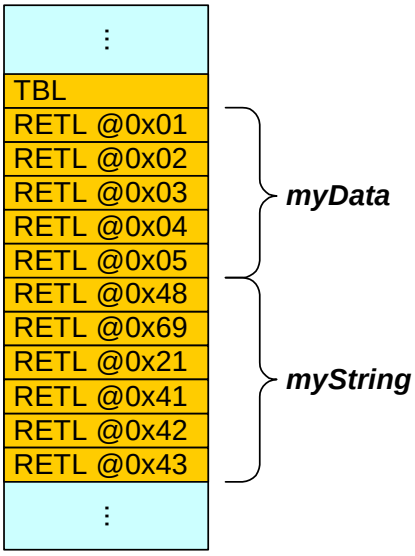
- *Use constant data type to store data into the ROM.*
- *Only global variable can be declared as "const" data type.*
- *The maximum size of a constant array variable is 255 bytes.*

Example:

```
const int myData[] = {1, 2, 3, 4, 5};

const char myString[2][3] = {
        “Hi!”,
        “ABC”
};
```

Program ROM:



NOTE

If the specified MCU does not support TBL instruction, a page has only one ROM data area (below 0x100); otherwise a page has a maximum of two ROM data areas.

7.8 Inline Assembler

The compiler has an in line assembler which allows you to enhance the functionality of your program.

7.8.1 Reserved Word

The reserved words for the inline assembler are**:**

```
_asm
{
      …… //write assembly code here
}
```

All the assembly instructions (in upper or lower case) of the EM78 series are supported.

NOTE

- *Users can declare variable in 0x10~0x1F when C Compiler doesn't use. See 7.1 for the detail.*
- *If user has to switch "rpage," "iopage," or "bank" in the inline assembly, the original "rpage," "iopage," or "bank" must be saved at the beginning and restored at the end of the inline assembly program section. Refer to Example 1 in the next section (Section 7.8.2).*
- *If users use 0x10~0x1F in inline assembler, compiler would not report warning or error message, but it may meet some unexpected errors.*
- *Users can't use _ASM to replace _asm.*
- *If users use register address, io control address or ram bank address directly, compiler don't know these registers, io control or rams used in assembly. Users have check it.*

7.8.2 Use of C Variable in the Inline Assembly

The Compiler allows you to access the C variable in the inline assembly as follows**:**

```
mov a, %<variable name>   //move variable value to ACC
mov a, @%<variable name>  //move address of variable to ACC
```

Example 1:

```
_asm
{
// Save procedure of rpage, iopage and bank register
     mov a,0x0
     mov %nbuf, a
     mov a, 0x04
     mov %nbuf+1, a
     bs 0x03, 7
     bs 0x03, 6    //Switch to other rpages
     ……
                   //Restore procedure of rpage, iopage and bank
     mov a, %nbuf //register
     mov 0x03, a
     mov a, %nbuf + 1
     mov 0x04, a
}
```

Example 2:

```
int temp;
temp=0x03;               //we suppose temp is at 0x21 of bank 0
_asm {mov a, %temp}      //move value 0x03 to ACC
_asm {mov a, @%temp}     //move address 0x21 to ACC
```

Example 3:

```
unsigned int temp_a @0x20: bank 0;
unsigned int temp_s @0x21: bank 0;
#define status 0x03;
void main()
```

```
{
_    asm
     {
        mov %temp_a, a   // ➔ mov 0x20, a
        mov a, status    // ➔ mov a, 0x03
        mov %temp_s, a   // ➔ mov 0x21, a
     }
}
```

7.9 Using Macro

You can use macro to control the MCU and shorten the program length.

NOTE

- *Use "#define" to declare a macro.*
- *Use "\" to join more than one line assembly codes.*
- *Do not add any character after "\" (even a block character is not allowed), otherwise an error will occur.*
- *Do not use constant as variable in macro. It will meet error.*

Example:

```
#define SetIO(portnum, var)     _asm {mov a, @var} \
                                _asm {iow portnum}

#define SetReg(reg, 3)          _asm {mov a, @3} \
                                _asm {iow portnum}
macro SetReg will meet error for using constant as argument.
```

7.10 Interrupt Routine in TCC1

To handle Interrupts, two things have to be taken into account**:**

1. **Interrupt Save Procedure:** the procedure to save some registers before executing a service routine. For instance, ACC, R3 should be saved in the EM78P458 as Interrupts occur. Only inline assembly is allowed under Interrupt Save Procedure.
2. **Interrupt Service Routine:** is the action to be taken for Interrupt.

NOTE

- *You may ignore the details on setting interrupts as WicePlus will handle all these tasks. However, you need to concentrate more on the interrupt service routine.*
- *The "page" instructions cannot allocate the same ROM space as interrupt service routine (see Section 3.6 .*

7.10.1 Interrupt Save Procedure

void _intcall <function name>_l(void) @<interrupt vector address>: low_int <interrupt vector number>

It should be noted that "_l" (for low level interrupt) must be added after function name.

7.10.2 Interrupt Service Routine

void _intcall <function name>(void) @int <interrupt vector number>

The <interrupt vector number> means that if there are many interrupt vectors in the MCU, the sequence 0, 1, 2, 3… is provided to separate each interrupt vectors.

The compiler will automatically combine the saved procedure and the service routine in the <interrupt vector number>. That is MCU will jump from interrupt save procedure to interrupt service procedure.

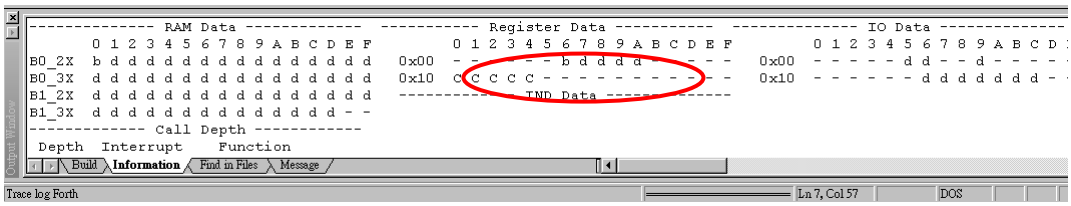
NOTE

- *Interrupt Save Procedure and Interrupt Service Routine cannot be assigned with parameters; otherwise the compiler will generate an error.*
- *Interrupt routine only supports one byte data operation (such as "int", "char"), otherwise the compiler will generate an error.*
- *Interrupt service routine doesn't support array calculation except save and restore common register inline assembly. If there is an array calculation in interrupt service routine, compiler will report error. But users can call a function with array calculation. Example 4 illustrates this.*
- *Under interrupt service routine, you can call other functions. But long data type's *, / and % operation are not allowed in the called function. Refer to Example 3 below.*
- *You must write an inline-assembly code to save some registers in the Interrupt Save Procedure per MCU type. For instance, ACC, R3, R4, and R5 should be saved in EM78R806B as interrupts occur. If the MCU supports hardware backup control for the interrupt routine, you may ignore saving the registers in the Interrupt Save Procedure. Please study the MUC spec about that. It is very important to switch to program page 0 at the end of Interrupt Save Procedure.*
- *Users have to confirm whether the bank of these saving address in interrupt save procedure is the same as the bank of restoring in interrupt service procedure.*
- *Users can't use these ram spaces that are used to backup ACC, R3, R4, R5 or other general purpose registers 0x10~0x1F.*

7.10.3 Reserved Common Registers Operation

Sixteen common registers (0x10~0x1f) are reserved for certain operation. When an interrupt occurs, it is strongly recommended that users to backup some common registers. After Compiling, WicePlus will tell users which registers have to backup from information window, Output window. In the picture below, users can see there are five C characters in the line of 0x10. These positions are 0x10, 0x11, 0x12, 0x13, 0x14. So users have to save these 5

common registers and restore them in interrupt service routine. That is users have to compile a time to know these message. Character C in the line 0x10 means C compiler occupied in other functions. Compiler will dynamic to use these in WicePlus 2.



Example 1:

```
Void _intcall INTERRUPT1_l(void) @0x08: low_int 0
{
     // backup ACC, R3, R4, R5
     _asm
     {
          MOV 0X1F, A
          SWAPA 0X4
          BS 0X4, 6    // switch to ram bank 3
          BS 0X4, 7
          MOV 0X3F, A
          SWAPA 0X3
          MOV 0X3E, A
          SWAPA 0X5
          MOV 0X3D, A
          PAGE @0X0    //or Use BC to switch program page 0 if the
     }                 //MCU doesn't support page instruction
}

void _intcall INTTERRUPT1(void) @int 0
{
     // backup C system
     _asm
     {
          MOV A, 0X10 // C system backup
          MOV 0X3C, A  // save 0x10~0x14 to 0x3C,0X3B, 0X3A, 0X39,
          MOV A, 0X11 //0X38 in bank 3 because switch to ram
          MOV 0X3B, A //bank 3 in _intcall INTERRUPT1_l
          MOV A, 0X12
          MOV 0X3A, A
          MOV A, 0X13
          MOV 0X39, A
          MOV A, 0X14
          MOV 0X38, A
     }
```

```
        // Write your code (inline assembly or C) here
        ……
        // restore C system
        _asm
        {
             BS 0X04, 6 // switch to ram bank 3 to restore correctly
             BS 0X04, 7
             MOV A, 0X3C //C system restore
             MOV 0X10, A
             MOV A, 0X3B
             MOV 0X11, A
             MOV A, 0X3A
             MOV 0X12, A
             MOV A, 0X39
             MOV 0X13, A
             MOV A, 0X38
             MOV 0X14, A
        }
        // restore ACC, R3, R4, R5 following backup C system
        _asm
        {
             SWAPA 0X3D //Users have to confirm whether in ram bank 3
             MOV 0X5, A  //or not. If not, have to switch to ram bank
             SWAPA 0X3E //3 to restore correctly
             MOV 0X3, A
             SWAPA 0X3F
             MOV 0X4, A
             SWAP 0X1F
             SWAPA 0X1F
        }
}
```

Example 2:

```
int nBuf[5];
void _intcall INTERRUPT2_l(void) @0x08: low_int 0
{
     // backup ACC, R3, R4, R5
     _asm
     {
          ……   //switch bank by users
     }
}
```

```
void _intcall INTERRUPT2(void) @int 0
{
     _asm //save registers
     {
          mov a, 0x10       //Compiler will switch bank auto
          mov %nBuf, a
          mov a, 0x11
          mov %nBuf + 1, a
          mov a, 0x12
          mov %nBuf + 2, a
          mov a, 0x13
          mov %nBuf + 3, a
          mov a, 0x14
          mov %nBuf + 4, a
     }
     // do what you want to do as interrupt occurred.
     ……
     _asm //restore registers
     {
          mov a, %Buf       //Compiler will switch bank auto
          mov 0x10, a
          mov a, %nBuf + 1
          mov 0x11, a
          mov a, %nBuf+2
          mov 0x12, a
          mov a, %nBuf + 3
          mov 0x13, a
          mov a, %nBuf + 4
          mov 0x14, a
     }

    _asm
     {
       …       //switch bank by users
     }
}
```

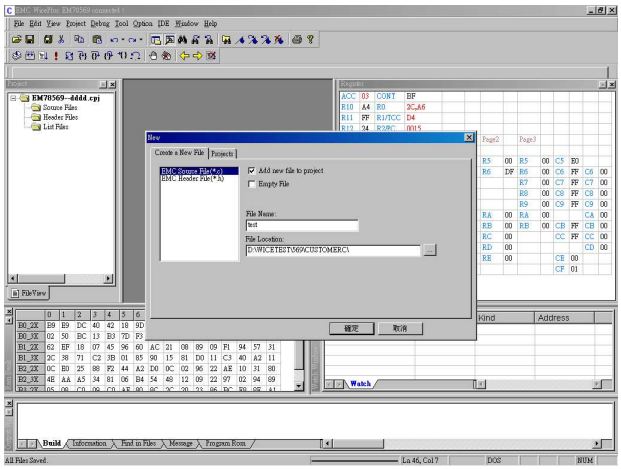
Example 3:

```
void _intcall INTERRUPT3_l(void) @0x08: low_int 0
{
     // backup ACC, R3, R4, R5
     _asm
     {
          ……
     }
}
void _intcall INTERRUPT3(void)@int 0
{
     long ans;
     ……
     ans = LongMult(0x1234, 0x5678 );
     ……
}
long LongMult(long a, long b)
{
     return (a * b);
          // multiple operation of long data type is NOT allowed!
```

Example 4:

```
unsigned int IRBR[6];
void abcd()
{
   int i,j;
   for(i=0;i<6;I++)
    {
       for(j=0;j<3;j++)
        {
           IRBR[i]= IRBR[i]>> 1;
        }
    }
}
void _intcall INTERRUPT3_l(void) @0x08: low_int 0
{
     // backup ACC, R3, R4, R5
     _asm
     {
          ……
     }
}
void _intcall INTERRUPT3(void)@int 0
{
     ……
     abcd();
     ……
}
```

WicePlus supply interrupt save procedure and interrupt service routine form in the file where the main function locates. After building a new project, users can adds a new file like below figure with main() function and interrupt save procedure and interrupt service routine. Please remember just one main() function could be in a C project. So next time when you want to add another new file in the project, you have to check Empty file box.



There is a very important difference about save and restore registers between Example 1 and Example 2. In example 2, compiler allots the locations where variable nBuf is. So compiler identifies nBuf variables in inline assembly. Compiler will check variables redefined or not. In example 1, if users declare some variables at 0x38, 0x39, 0x3A, 0x3B, 0x3C in bank 1 in other location in the project, compiler doesn't check reuse. That's compiler can't report error. But error will occur during ICE running.

7.11 Interrupt Routine in TCC2

Interrupt Routine with TCC2 is easier than with TCC1. Also it really reduces rom space. In TCC2, to handle interrupts, just three have to be taken into account and one is new**:**

1. **Interrupt Save Procedure:** the same as with TCC1, the procedure to save some registers before executing a service routine. Now many new IC save important register like ACC, R3 or R4 as interrupt occurs and restore them before it quits interrupt service. Also TCC2 can save and restore these registers that WicePlus supplies. If not, WicePlus will supplies these form when a new one file added in a new project. If not only one interrupt vectors in one IC, users have to save register 0x2 (PC) to ACC. TCC2 use the value of PC to know which interrupt source.
2. **Interrupt Service Routine:** is the action to be taken for Interrupt. Don't care how many interrupt vectors, users have to write interrupts service code in it and use switch case or if-else if to know which interrupt vector source is if need.
3. **Global Interrupt Vector Index variable (IntVecIdx):** Users have to declare IntVecIdx as global integer like "extern int IntVecIdx;". IntVecIdx will occupy 0x10. So users can't declare or use 0x10 anywhere in project. Compiler doesn't tell you if you declare or use but it will run a wrong result.

7.11.1 Interrupt Save Procedure

void _intcall <function name>_I(void) @<interrupt vector address>: low_int <interrupt vector number>

Interrupt save procedure format is the same as in TCC1. But users have to write "MOV A,0x2" first in inline assembly code.

7.11.2 Interrupt Service Routine

void _intcall <function name>(void) @int

The <interrupt vector number> is skipped in TCC2 interrupt service routine and just one interrupt service routine function.

In example 1,not only one interrupt vector in EM78P510N, write interrupt service each case relative to interrupt vector. Users can mark case and interrupt save procedure if they aren't used.

In example 2, although just one interrupt vector in EM78569, users can't use 0x10 but don't declare IntVecIdx.

In example 3, in EM78567, users can't use 0x10 but don't declare IntVecIdx. Hardware don't save and restore and register.

Users have to note that some IC in restoring inline assembly use reti in the first line. It tells compiler to restore automatically from this line. Compiler will restore automatically after inline assembly if no reti. Reti have to put the first order in restoring inline assembly.

7.11.3 Reserved Common Registers Operation

Compiler save common registers (0x11~0x1F) compiler uses. Note the use of reti in restoring inline assembly in some IC.

Example 1:
EM78P510N: Hardware save and restore ACC, R3 and R5 in interrupt action. Not only one interrupt vector

```
extern int IntVecIdx; //occupied 0x10:rpage 0
…
void _intcall AllInt(void) @ int
{
  switch(IntVecIdx)
  {
    case 0x4://write interrupt vector 0x3 in this case
    break;
```

```
    case 0x7://write interrupt vector 0x6 in this case
    break;

    case 0xA://write interrupt vector 0x9 in this case
   break;

    case 0xD://write interrupt vector 0xC in this case
    break;

    case 0x10://write interrupt vector 0xF in this case
    break;

    case 0x13://write interrupt vector 0x12 in this case
    break;

    case 0x16://write interrupt vector 0x15 in this case
    break;

    case 0x19://write interrupt vector 0x18 in this case
    break;

    case 0x1C: //write interrupt vector 0x1B in this case
    break;

  }
/* User also can use if-else if
  if(IntVecIdx==0x4)
  {
  }else if(IntVecIdx==0x7)
  {
  } else if(IntVecIdx==0xA)
  {
  } else if(IntVecIdx==0xD)
  {
  } else if(IntVecIdx==0x10)
  {
```

```
  } else if(IntVecIdx==0x13)
  {
  } else if(IntVecIdx==0x16)
  …

*/
}

void _intcall TCC_l(void) @ 0x03:low_int 0
{
 _asm{MOV A,0x2}; //using LJMP to interrupt service procedure
}
void _intcall ExtInt_l(void) @ 0x06:low_int 1
{
 _asm{MOV A,0x2};
}
void _intcall WatchTime_l(void) @ 0x09:low_int 2
{
 _asm{MOV A,0x2};
}
void _intcall Time1_l(void) @ 0x0C:low_int 3
{
 _asm{MOV A,0x2};
}
void _intcall Time2_l(void) @ 0x0F:low_int 4
{
 _asm{MOV A,0x2};
}
void _intcall ADC_l(void) @ 0x12:low_int 5
{
 _asm{MOV A,0x2};
}
void _intcall UART_l(void) @ 0x15:low_int 6
{
 _asm{MOV A,0x2};
}
void _intcall SPI_l(void) @ 0x18:low_int 7
```

```
{
 _asm{MOV A,0x2};
}
void _intcall LVD_l(void) @ 0x1B:low_int 8
{
 _asm{MOV A,0x2};
}
```

Example 2:
EM78569: Hardware save and restore ACC, R3 and R5 in interrupt action. WicePlus will tell Compiler to save and restore R4. One interrupt vector

```
void main()
{
  _asm{MOV A,@0x10 //enable bit 4 ,CONT
       CONTW
  }
}
void _intcall interrupt(void) @ int
{
 // Write your code (inline assembly or C) here
}
void _intcall interrupt_l(void) @ 0x08:low_int 0
{
  _asm{PAGE @0x0} //it's dangerous to skip this line.
}
```

Example 3:
EM78567: Hardware doesn't save and restore any register and Compiler just can save and restore R3 and R4. One interrupt vector

```
void _intcall interrupt(void) @ int
{
 // Write your code (inline assembly or C) here


   //restore ACC and R5
   _asm {
      reti      //tell compiler to recover common registers here
      SWAPA 0X1E
```

```
        MOV 0X5,A
        SWAP 0X1F
        SWAPA 0X1F
     }
}
void _intcall interrupt_l(void) @ 0x08:low_int 0
{
  //save ACC and R5
  _asm {
     MOV 0X1F,A
     SWAPA 0X5
     MOV 0X1E,A
     PAGE @0X0
   }
}
```

Chapter 8

To grip Tiny C Compiler quickly

In this chapter, we'll introduce yours to control C compiler quickly. Please execute WicePlus. Users can double click WicePlus icon to do that.

8.1 Create a new project

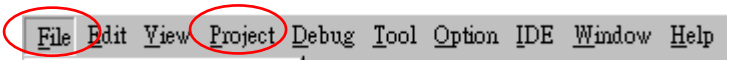


Fig. 8-1 Main menu

Click New in File (Fig. 8-2) or in Project (Fig. 8-3) popup menu from menu (Fig. 8-1) to call New dialog (Fig. 8-4). Then we can select folder and type project name. Press OK button to create project. Don't type extend file name. WicePlus will add extend file name cpj for the project. In this case, we select EM78P468N for our target MCU during connection.

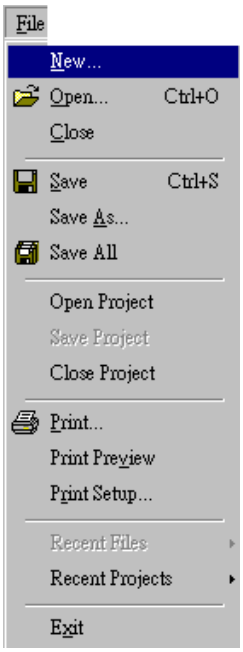


Fig. 8-2 File popup menu

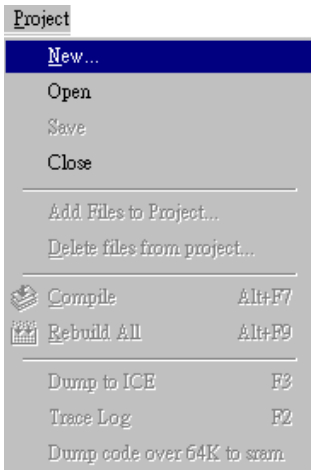


Fig. 8-3 Project popup menu

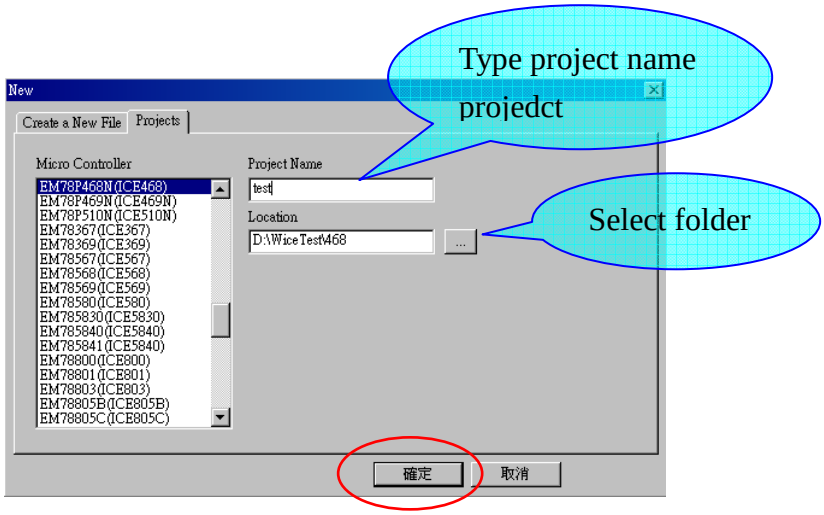


Fig. 8-4 Use New dialog to create a new C project

Now target MCU and the project name we create appear in Project window, Fig. 8-5.

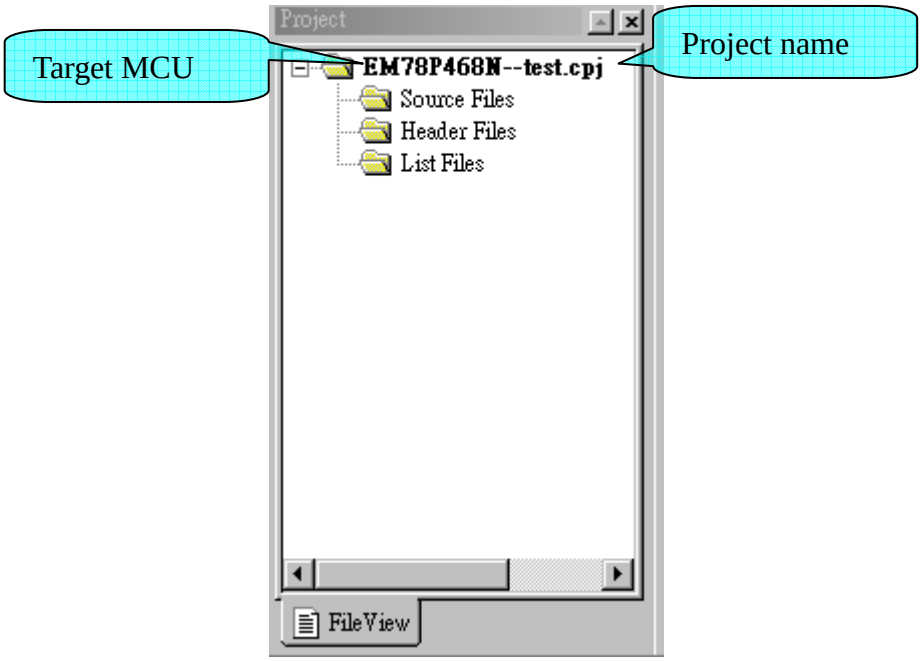
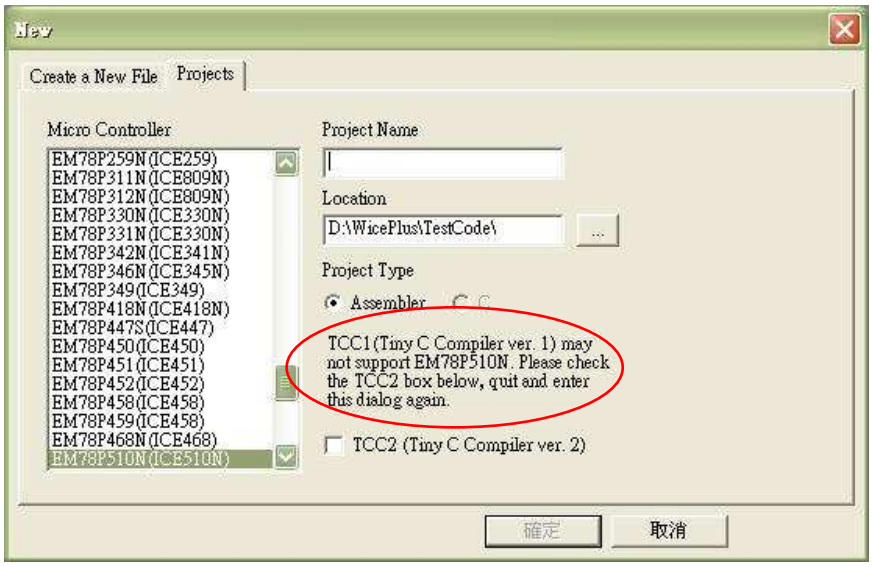
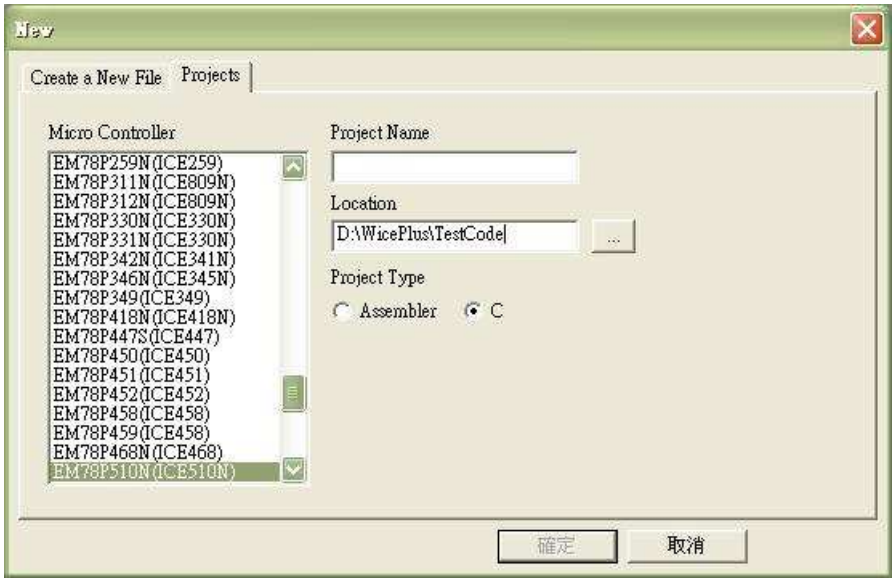


Fig. 8-5 MCU and project name in Project window

Supplement



This dialog will show some information about C Compiler in the red line circle. The information tells us now C Compiler is in TCC1 (Tiny C Compiler ver.1), and TCC1 doesn't support this IC, such as EM78P510N. Follow the information and check TCC2 check box, C Compiler will switch to TCC2 (Tiny C Compiler ver.2). Quit the dialog by clicking Cancel button, then re-enter the dialog, WicePlus will scan TCC2 support the selected IC or not. If support, user can create a TCC2 project.



8.2 Add a New C file to the project

Call New dialog like sec 8.1. This time New dialog appears and switch automatically to Create a New File sheet in New dialog, Fig. 8-5. WicePlus will supplies basic Tiny C compiler main() file and interrupt save procedure and interrupt service routine frame if select extend file name as c in left combo box and leave Empty File check box uncheck. We'll introduce the frame later. Add new file to project box checked is default. Now type the file name for the first file to add into the project. Don't type extend file name. WicePlus will add extend file name automatically. Press OK button to add the c first file to the project.

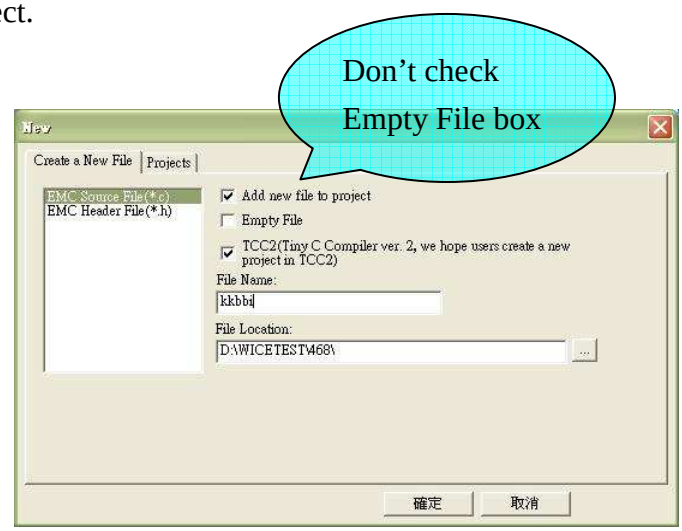


Fig. 8-5 Add the first c file with main() and interrupt save procedure and service routine frame

8.3 Add a new second file or head file to the project

This time we have to check Empty File box in Create a New file sheet in New dialog.

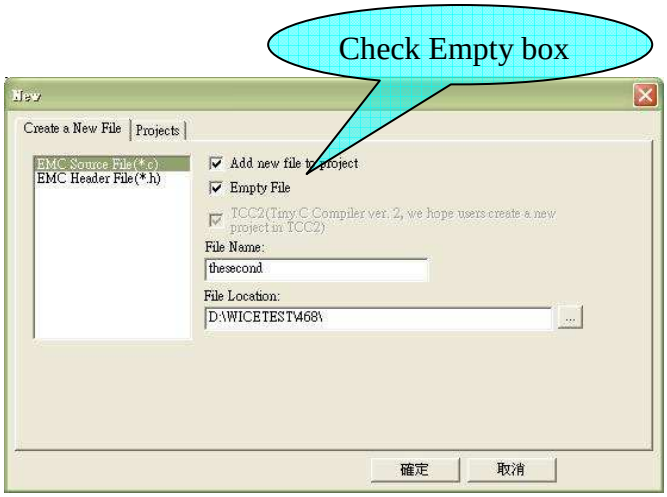


Fig. 8-6 Add a new second file or head file to the project

Type file name and press OK button WicePlus will create a new and empty c file. Head file will be added to project by empty and don't care about Empty File check box. Now we create a project with some c files and one head file, such as Fig. 8-7.

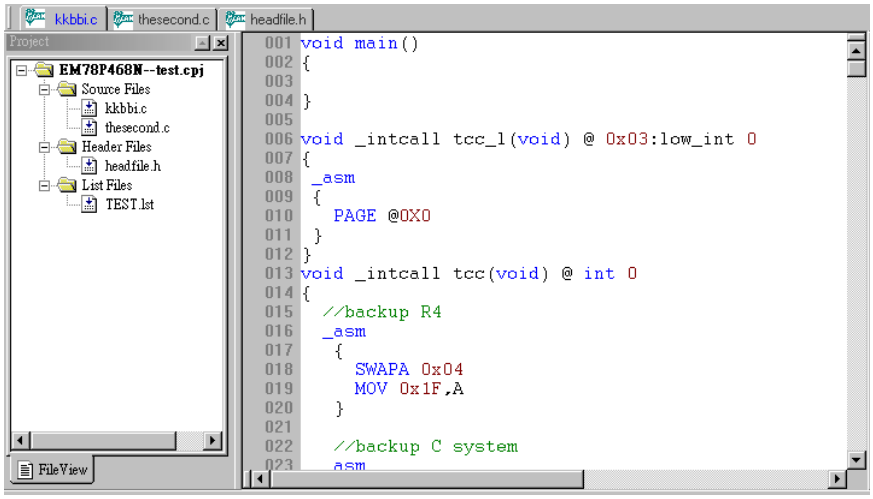


Fig. 8-7 We create a project with some c files and one head file.

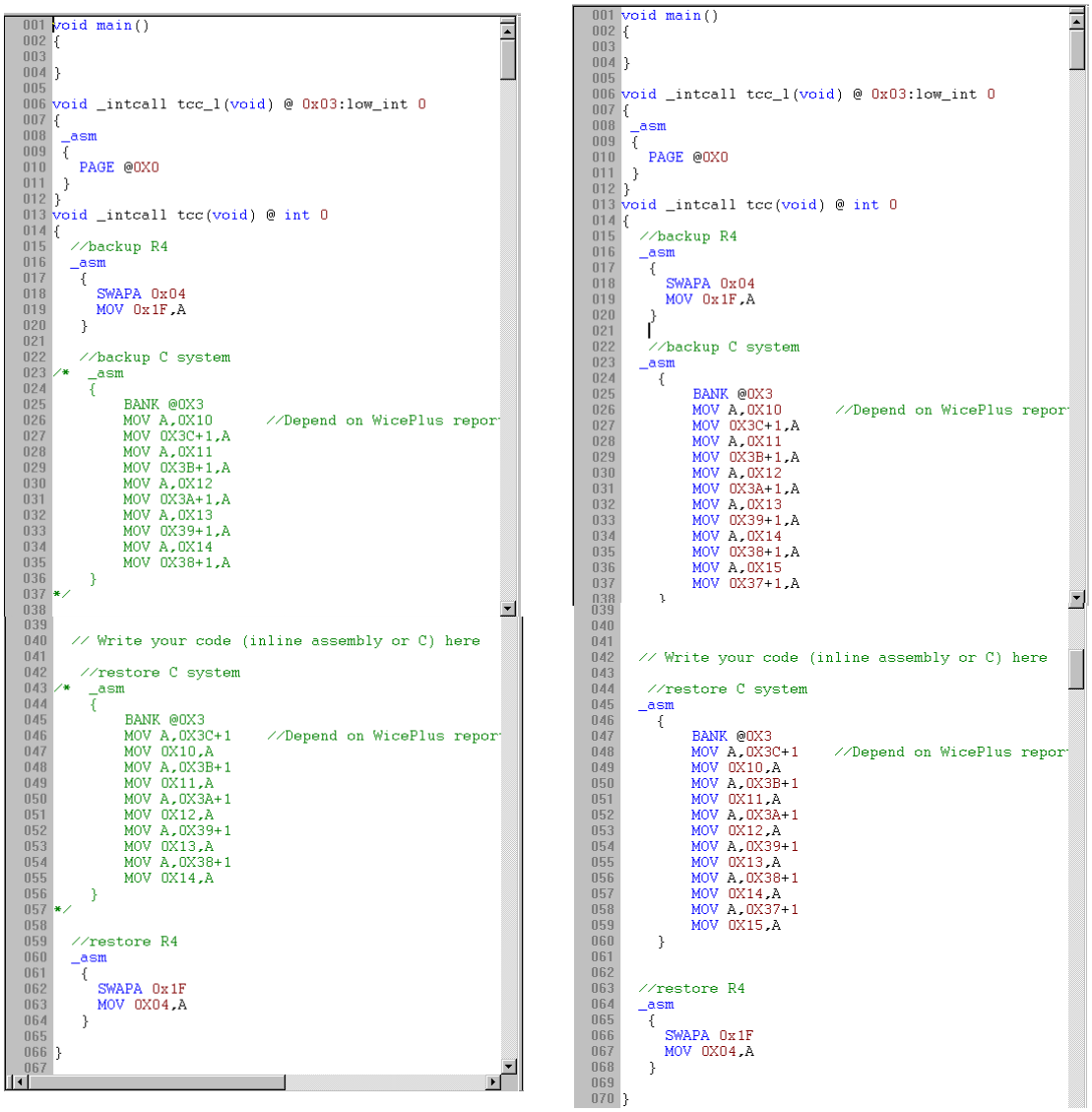
8.4 The main() and interrupt save and service frame that WicePlus supplies

Fig. 8-8 shows part of the frame. Just only one main() function can exist in C project.

Usually users don't need to modify save and restore about R4, R3, ACC or R5 in the file that WicePlus supplies. But for safety reason we suggest users to check if these save or restore location are at the same ram bank. In this case, MCU EM78P468N hardware saves ACC and R3 before execute interrupt service routine and restore these two registers when interrupt service routine is completely. (Reference to Sec. 4.8 Interrupt, EM78P468 8-Bit Microcontroller Product Specification). So we just have to save and restore R4 into 0x1F. For 0x1F is common register and belong to global field, we don't need to switch ram bank. We use swap instruction to avoid flag bit influenced. If we save R4 to 0x3F in ram bank 3, we have to restore 0x3F in ram bank 3 to R4. If we backup incorrectly, MCU may execute wrong PC (program counter) because R4 in EM78P468N control program page. We use inline assembly to do save and restore process (Ref. Sec. 7.8 Inline Assembler). Save and restore ACC, R3, R4 or R5 we call save and restore MCU hardware.

In addition to save and restore MCU hardware, some of cases we have to save and restore C system. Some calculations outside interrupt service routine in C compiler don't only use ACC or the declared register or ram but some of common registers 0x10~0x1F. And in interrupt service routine also uses these common registers. We have to confirm these common registers' value are the same before running into interrupt save procedure and leaving interrupt service routine. So we have to save and restore these common registers correctly. C compiler will tell us after compiling succeed. Please see Sec. 8. 5. In the file frame, we suppose 0x10~0x14 common registers have to save and restore. Please remark these two backup C system and restore C system inline assembler code if need. If C compiler tell us more common registers over 0x10~0x14 have to save and restore, just add code like MOV A, 0x15 and MOV 0x37+1, A after backup 0x14 in backup C system inline assembler and MOV A, 0x37+1 and MOV 0x15, A after restore 0x14 in restore C system inline assembler.

Users write interrupt service code between backup C system and restore C system, such as //Write your code (inline assembly or C) here in the case.



```
void main()
{

}

void _intcall tcc_1(void) @ 0x03:low_int 0
{
 _asm
 {
   PAGE @0X0
 }
}
void _intcall tcc(void) @ int 0
{
  //backup R4
  _asm
   {
     SWAPA 0x04
     MOV 0x1F,A
   }

  //backup C system
/*  _asm
    {
        BANK @0X3
        MOV A,0X10      //Depend on WicePlus repor
        MOV 0X3C+1,A
        MOV A,0X11
        MOV 0X3B+1,A
        MOV A,0X12
        MOV 0X3A+1,A
        MOV A,0X13
        MOV 0X39+1,A
        MOV A,0X14
        MOV 0X38+1,A
    }
*/


  // Write your code (inline assembly or C) here

  //restore C system
/*  _asm
    {
        BANK @0X3
        MOV A,0X3C+1    //Depend on WicePlus repor
        MOV 0X10,A
        MOV A,0X3B+1
        MOV 0X11,A
        MOV A,0X3A+1
        MOV 0X12,A
        MOV A,0X39+1
        MOV 0X13,A
        MOV A,0X38+1
        MOV 0X14,A
    }
*/

  //restore R4
  _asm
   {
     SWAPA 0x1F
     MOV 0X04,A
   }
}
```

```
void main()
{

}

void _intcall tcc_1(void) @ 0x03:low_int 0
{
 _asm
 {
   PAGE @0X0
 }
}
void _intcall tcc(void) @ int 0
{
  //backup R4
  _asm
   {
     SWAPA 0x04
     MOV 0x1F,A
   }

   //backup C system
  _asm
    {
        BANK @0X3
        MOV A,0X10      //Depend on WicePlus repor
        MOV 0X3C+1,A
        MOV A,0X11
        MOV 0X3B+1,A
        MOV A,0X12
        MOV 0X3A+1,A
        MOV A,0X13
        MOV 0X39+1,A
        MOV A,0X14
        MOV 0X38+1,A
        MOV A,0X15
        MOV 0X37+1,A
    }


  // Write your code (inline assembly or C) here

   //restore C system
  _asm
    {
        BANK @0X3
        MOV A,0X3C+1    //Depend on WicePlus repor
        MOV 0X10,A
        MOV A,0X3B+1
        MOV 0X11,A
        MOV A,0X3A+1
        MOV 0X12,A
        MOV A,0X39+1
        MOV 0X13,A
        MOV A,0X38+1
        MOV 0X14,A
        MOV A,0X37+1
        MOV 0X15,A
    }


  //restore R4
  _asm
   {
     SWAPA 0x1F
     MOV 0X04,A
   }
}
```

Fig. 8-8 WicePlus supplies C main and interrupt frame (left) and remark and add 0x15 save and restore (right).

8.5 Developing your project, especially about interrupt

Now we can write our c code to develop our product. In the case we wake up MCU by port 6 (Users can download the example from our Web http://www.emc.com.tw/database/Sa2/Gp/Dt/EM78P468C_Example.zip, file Emp7p468_pin wakeup.c) with Counter 1 underflow interrupt with interrupt vector 0xC. So we write some initial code and our interrupt service code like Fig. 8-9 and Fig. 8-10. Now we active one of c files and use Alt+F7 to compile

the file to see if there is errors in the file. Alt+F7 just compile the active file. During developing we use Alt+F7 to compile c file one by one to save compiling time. After no errors in all c files in the project occur we use rebuild All (Alt+F9) to compile and link our object code and create execute file (cds). If rebuild All succeeds C Compiler will report many important and useful messages in Information sheet in Output Window. Fig. 8-11 shows them.

Fig. 8-9 Write initial code

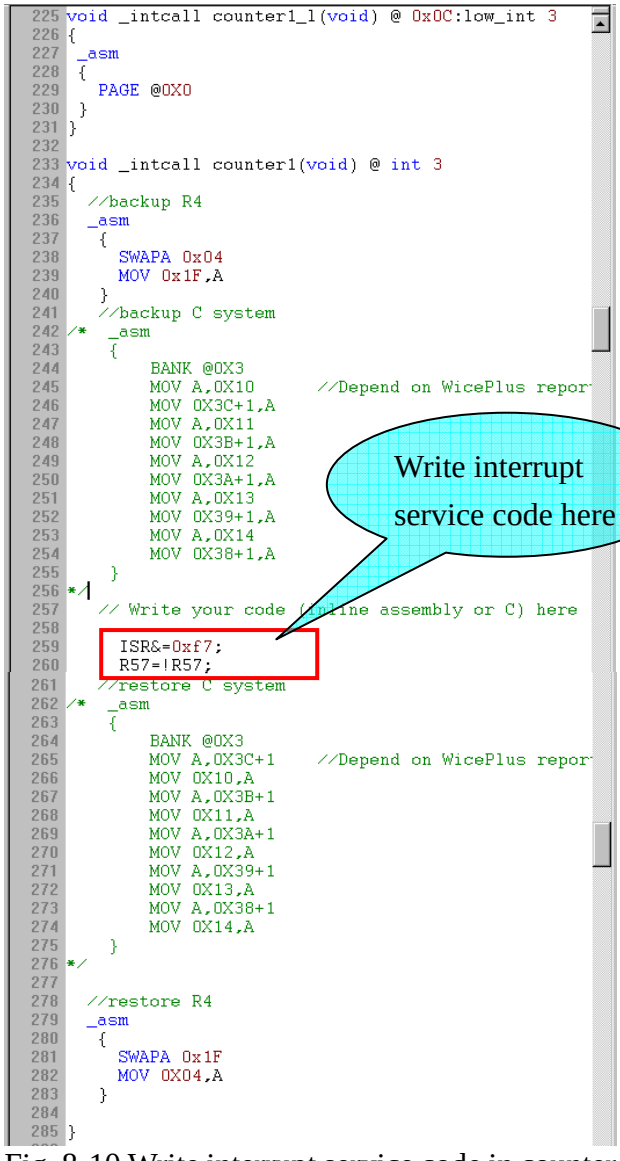


Fig. 8-10 Write interrupt service code in counter 1 interrupt (right)

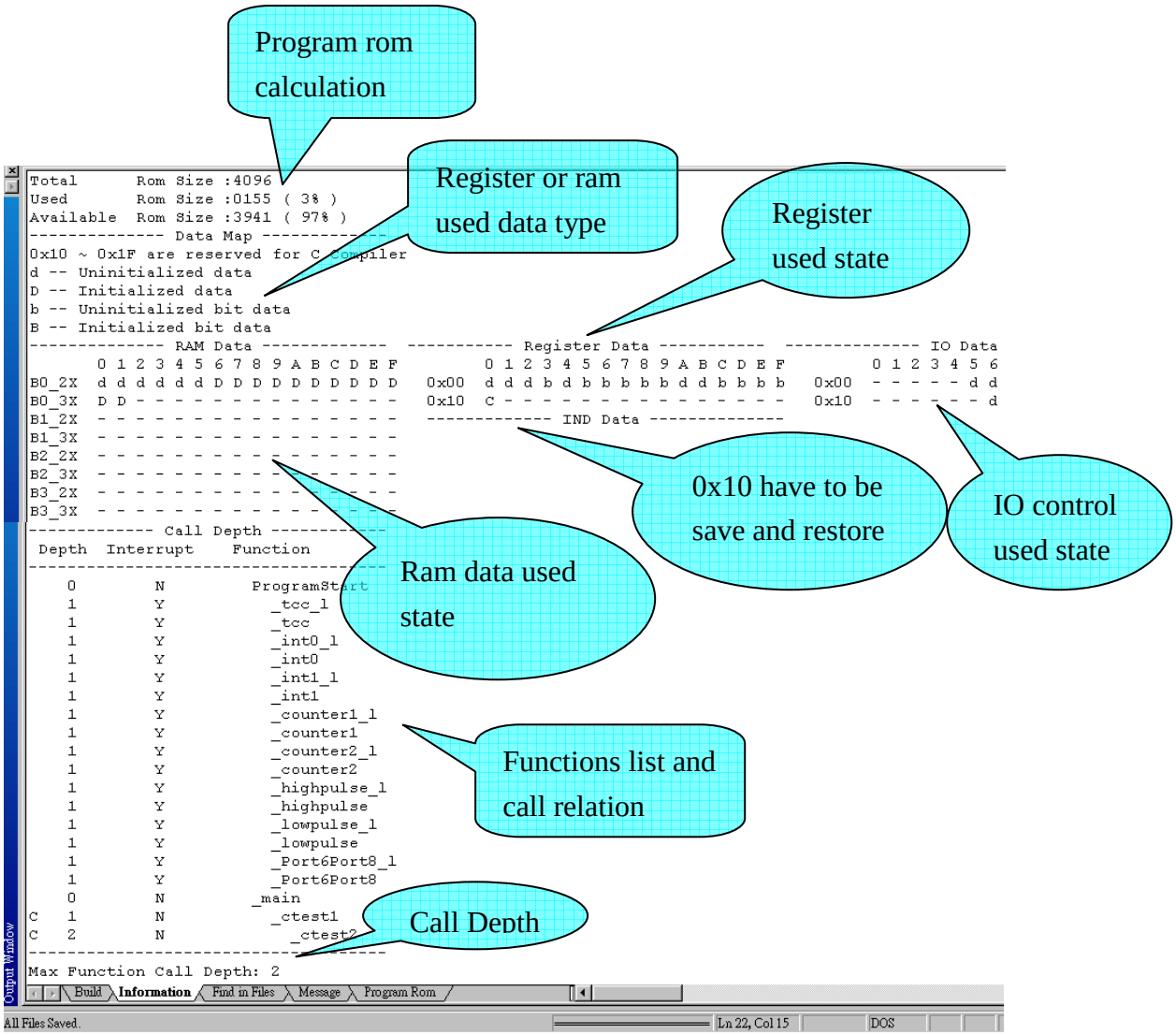


Fig. 8-11 Useful messages after rebuild all successfully

Fig. 8-11 tells us which common registers we must save and restore. C char is located at 0x10. So we modify C system save and restore like Fig. 8-12.

```
233 void _intcall counter1(void) @ int 3
234 {
235   //backup R4
236   _asm
237    {
238      SWAPA 0x04
239      MOV 0x1F,A
240    }
241    //backup C system
242   _asm
243     {
244        BANK @0X3
245        MOV A,0X10       //Depend on WicePlus repor
246        MOV 0X3C+1,A
247     }
248
249    // Write your code (inline assembly or C) here
250
251      ISR&=0xf7;
252      R57=!R57;
253    //restore C system
254   _asm
255     {
256        BANK @0X3
257        MOV A,0X3C+1     //Depend on WicePlus repor
258        MOV 0X10,A
259     }
260
261   //restore R4
262   _asm
263    {
264      SWAPA 0x1F
265      MOV 0X04,A
266    }
267
268 }
```

Fig. 8-12 Save and restore C system correctly

After save and restore hardware system and C system correctly, we can debug or run our project.

Chapter 9
C Compiler debug tips

9.1 Speed Up Debug

During in C environment, User can press Speed Up Debug button to improve debugging speed with Step Into (F7).

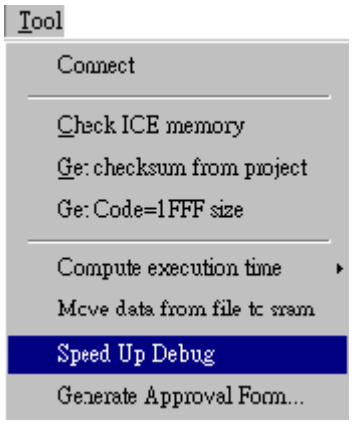


Fig. 9.1 Speed Up Debug

If users don't press Speed Up Debug button, WicePlus runs Step Into function (F7) by program counter (PC) to program counter (PC) just as in assembly environment. But if Speed Up Debug pressed, WicePlus run to at first assembly code address corresponding to a C source line except some call function.

For example blow C source is at the left and the corresponding assembly codes are at the right. The next Step into (F7) will run from program counter 0x29 to 0x30 in assembly code without stop corresponding to line 2855 in C code.

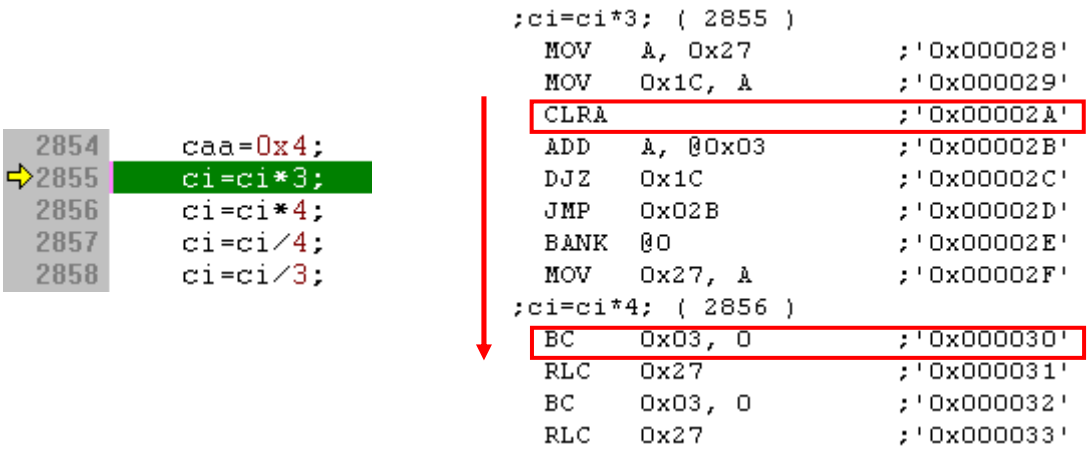


Fig. 9.2 Speed Up debug in C Step into (F7)debug mode

Please note some situations are right. For example, if the C code is ci++ in line 2856, so in Speed Up Debug after step into (F7) line 2588, the value of ci will increase 1.

9.2 View corresponding Assembly code in C environment

After first dump code to ICE (F3) in C environment, users can press Assembly code like figure x. After that during debugging, users can see C and assembly code fully occupy in Edit window. Users have to active and set focus on C file if you want to press any button in menu.

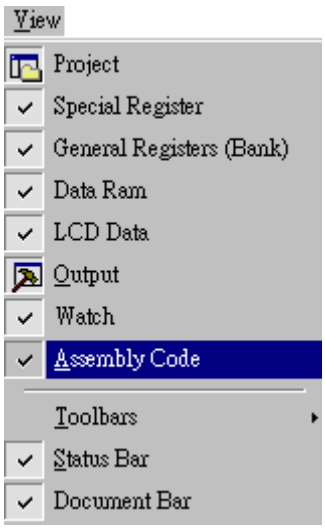


Fig. 9.3 Press Assembly code button in View

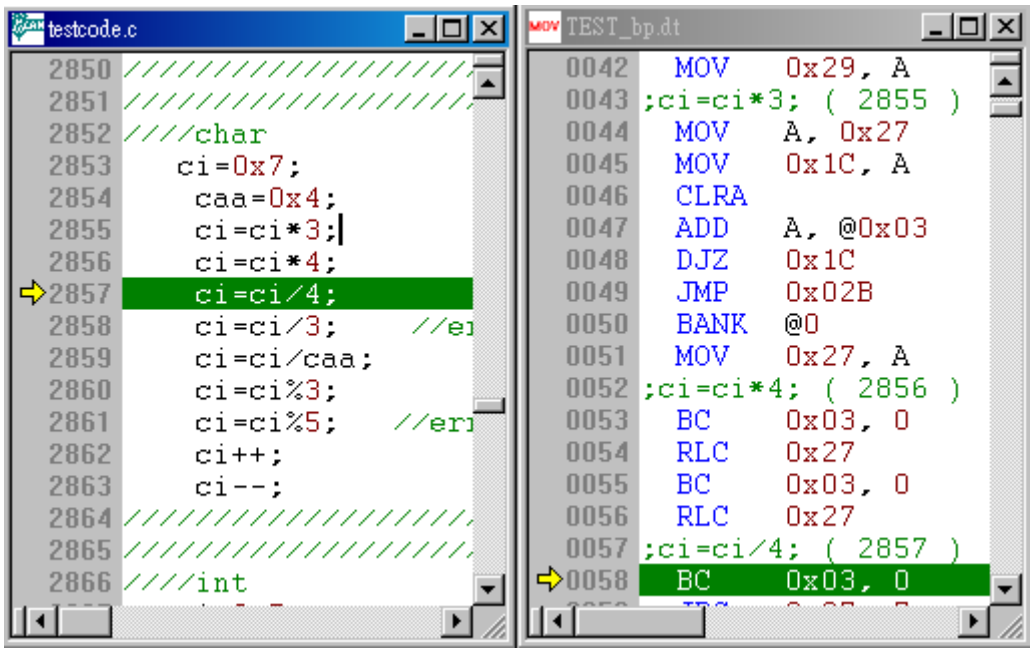
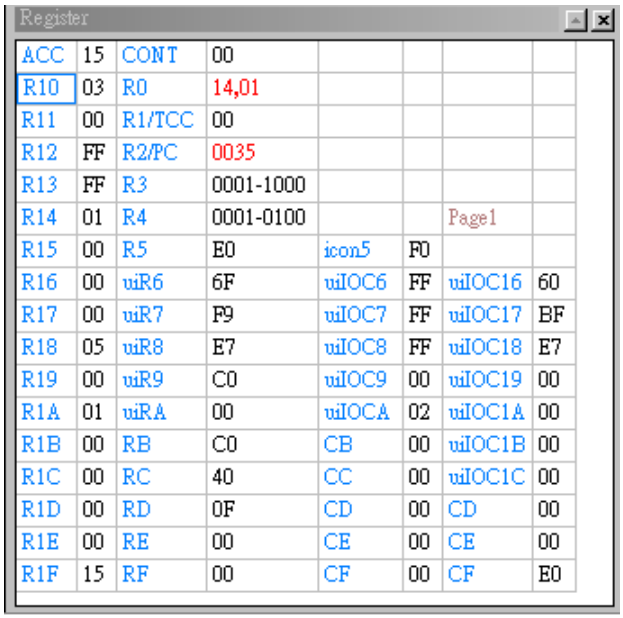


Fig. 9.4 Assembly code in C debug mode and focus on C file.

Users have to active and set focus on assembly file if you want to press “StepInto”,“StepOver”,”StepOut”,”Go”,”FreeRun”,”Reset”,”GotoCursor”, “insert/Remove Breakpoint” buttons.

9.3 Watch Defined variable in Register Window

Check Show defined label in Register window in view setting dialog after dumping, those variables declared in register field will show in register window. Registers name don’t show in default but users define.


Register
ACC 15 CONT 00
R10 03 R0 14,01
R11 00 R1/TCC 00
R12 FF R2/PC 0035
R13 FF R3 0001-1000
R14 01 R4 0001-0100 Page1
R15 00 R5 E0 icon5 F0
R16 00 uiR6 6F uiIOC6 FF uiIOC16 60
R17 00 uiR7 F9 uiIOC7 FF uiIOC17 BF
R18 05 uiR8 E7 uiIOC8 FF uiIOC18 E7
R19 00 uiR9 C0 uiIOC9 00 uiIOC19 00
R1A 01 uiRA 00 uiIOCA 02 uiIOC1A 00
R1B 00 RB C0 CB 00 uiIOC1B 00
R1C 00 RC 40 CC 00 uiIOC1C 00
R1D 00 RD 0F CD 00 CD 00
R1E 00 RE 00 CE 00 CE 00
R1F 15 RF 00 CF 00 CF E0

9.4 Reduce cods size in some cases

Define signed variable and unsigned variable are not the same. Use unsigned variable will reduce program code size.

Appendix A
Assembly Error Message

Error messages are categorized into the 4 classes which are class M, class A, class L, and class D. Any message of class M pops out when main program is executed incorrectly. Any message of class A is on the list if syntax errors, for instance, occur during assembling. Any message of class L is showed in the list if there are any linking errors occurred during a project built or rebuilt. Any message of class D describes the errors of debugging program.

A.1 Class M: main program errors.

a. "error M001: Numbers of opening editor windows are over limitation."
Reason: Too many opening windows to open one more.
Solution: To close some unused editor windows.

b. "error M002: Memory is not enough to allocate Editor Window."
Reason: System has not enough memory to allocate editor window.
Solution: To close some unused editor windows or unexecuted application programs in system.

c. "error M003: File: [filename] is existed."
Reason: The file name has been created already, so the same one can not be created again.
Solution: Rename and save again.

d. "error M004: File: [filename] can not be created."
Reason: The application is notified that the file cannot be created by O.S.
Solution: To check whether the disk is full or not and the system is stable or not.

e. "error M005: One project has been opened."
Reason: If one project has been opened in WICE, then users can not open another project.
Solution: To close the opening project, and then open another project.

f. "error M006: Project: [filename] can not be created."
Reason: The application is notified that the project cannot be created by O.S.
Solution: To check whether the project is empty or not.

g. "error M007: The file [filename] has already existed in the project."
Reason: The file has been included in the project.
Solution: To stop trying to add the file into the project again.

h. "error M008: File: [filename] can not be saved."
Reason: The file can not be saved in the disk.
Solution: To check whether the project is empty or not.

i. "error M009: The project: [filename] is not the format of EMC project file."
Reason: The content of the project is not the EMC project format.

Solution: To create a new project, and add the file to project.

j. **"error M010: The file: [filename] does not exist."**
Reason: The file can not be found in the directory.
Solution: To check whether the file exists or not.

k. **"error M011: The File: [filename] can not be opened."**
Reason: The file can not be opened by O.S.
Solution: To check whether the file is existed or not.

l. **"error M012: The file: [filename] exceeds [number]k of the max size [number]k."**
Reason: The file size is over the maximum size of buffer which editor is allocated.
Solution: To divide the file into two or more.

m. **"error M013: The copy size: [number] exceeds [number]k of the max size [number]k."**
Reason: The size of copy is over the maximum size of copy buffer.
Solution: To reduce content of the copy.

n. **"error M014: Memory can not be allocated."**
Reason: System can not allocate more memory to use.
Solution: To close unused editor windows or unexecuted application programs in system.

o. **"error M015: Over 250 characters of a line. "**
Reason: The maximum number of characters is 250 at a line in editor.
Solution: To divide a line into two or more.

p. **"error M016: The extension name of the active file [filename] is not .dt or .asm."**
Reason: In this action, the extension name of the active file must be “DT” or “ASM”.
Solution: To change the active file to another file with the extension name, “DT” or “ASM”.

q. **"error M017: No file to be assembled."**
Reason: Can not find a file to be assembled.
Solution: To select a file to be assembled.

r. **"error M018: Project file must be created."**
Reason: No project is available.
Solution: To open or create a project.

s. **"error M019: The editor number is over the max. editor number."**
Reason: The numbers of editors are limited.
Solution: To close some of unused editor windows.

t. **"Error M020:No active editor window."**
Reason: The editor window is not selected now.
Solution: To choose editor window.

u. **" Error M021: Text field must be input by characters."**
Reason: The text field can’t be empty.
Solution: To input character.

A.2 Class A: assembler errors.

a. **"error A001: Can not find the [filename] file."**
Reason: The file can not be found in the directory.

Solution: To check whether the file exists or not in the directory.

b. **"error A002: Main and subroutine programs can not define [label name] local label."**
   Reason: The local label (the preceding sign of label is "$") can not be defined in the main and subroutine programs.
   Solution: To remove the local label from the main or subroutine program.
c. **"error A003: The syntax form should be: operation [operand][,operand]."**
   Reason: The syntax form of statements must be "operation [operand][,operand]".
   Solution: To correct error of the line.
d. **"error A004: The label [label name] is redefined."**
   Reason: The label is defined more than once.
   Solution: To redefine the label name.
e. **"error A005: The EQU syntax is: label EQU operand."**
   Reason: The EQU syntax is error.
   Solution: To correct error of the line.

f. **"error A006: The INCLUDE nest depth is over 256."**
   Reason: The maximum depth of using "INCLUDE" is 256.
   Solution: To reduce the depth of "INCLUDE".
g. **"error A007: The IF conditional expression is error."**
   Reason: The expression of "IF conditional" is error.
   Solution: To correct the expression of "IF conditional".
h. **"error A008: Attempt to divide by zero."**
   Reason: The expression of number is divided by zero.
   Solution: To modify the expression.
i. **"error A009: The assembler does not support floating point."**
   Reason: The assembler number system does not support floating point.
   Solution: To change the floating point to integer.
j. **"error A010: The symbol [symbol name] is not defined."**
   Reason: The symbol is not defined.
   Solution: Please define the symbol.
k. **"error A011: The macro name [macro name] is redefined."**
   Reason: The macro name is defined more than once.
   Solution: To change the macro name to isolate name.
l. **"error A012: The parameter name [parameter name] is the same as label."**
   Reason: The parameter name is the same as label.
   Solution: To redefine the parameter name.
m. **"error A013: The parameter name [parameter name] is the same as another one."**
   Reason: The parameter name must be different from another one in the macro definition.
   Solution: To redefine the parameter name.
n. **"error A014: The number of actual parameter does not match with formal parameter."**
   Reason: The number of actual parameter is not matched with the number of formal parameter.
   Solution: To change the number of actual parameter or formal parameter.
o. **"error A015: The parameter number [number] does not exist."**

Reason: The position of actual parameter is not defined.
Solution: To change the number of actual parameter.

p. **"error A016: The external symbol [symbol name] is the same name as defined label."**
Reason: The external symbol name is the same as internal defined label.
Solution: To redefine the internal defined label.

q. **"error A017: Address of ORG is error."**
Reason: The address of ORG defined is error.
Solution: To redefine the ORG address.

r. **"error A018: MACEXIT can not be set outside macro."**
Reason: The "MACEXIT" instruction can not be set outside the macro definition.
Solution: To remove the "MACEXIT" instruction.

s. **"error A019: Parameter must be string variable."**
Reason: The formal parameter must be defined as string.
Solution: To change the formal parameter to string.

t. **"error A020: Memory can not be allocated."**
Reason: O.S. can not allocate extra memory.
Solution: To close the editor window which is not used currently or close application which is not used at present.

u. **"error A021: The source statements exceed [number] lines."**
Reason: The line number of source file is over the system default limit.
Solution: To divide the program into two or more small programs.

v. **"error A022: The tree is error because of parser error."**
Reason: The syntax of the program line is error.
Solution: To rewrite the program line.

w. **"error A023: [allocated memory type] memory fault when memory is allocated."**
Reason: O.S. can not allocate extra memory.
Solution: To close the editor window currently not used or close application currently not used.

x. **"error A024: Setting assembler variable must be integer value."**
Reason: The result of expression in the right side of "SET" instruction must be integer.
Solution: To change the expression.

y. **"error A025: Address of ORG must be integer value."**
Reason: The address of ORG must be integer.
Solution: To rewrite the address expression of "ORG" instruction.

z. **"error A026: The [number]address of PC is out of [number] ROM size."**
Reason: The address of program counter is over the program ROM size.
Solution: To reduce the code size.

aa. **"error A027: The assembler exceeds max. pass [number]."**
Reason: The pass of assembler is over the maximum default pass.
Solution: To check the assembler errors.

bb. **"error A028: The operand [number] value dose not include the valid data. "**
Reason: The operand value is invalid.
Solution: To change the operand value.

cc. **"error A029: The [number] address is conflicted."**
Reason: The program address is conflicted with another address.
Solution: To use function of "[Edit]->[Find]" menu to find the address in the list file; then to update the address of the program position.

dd. "error A030: The file [file name] can not be opened."
Reason: The file can not be opened.
Solution: To check whether the file exists or not.

ee. "error A031: The configure file read error."
Reason: The format of the configuration file is not correct.
Solution: To setup WICE software again.

ff. "error A032: The file [file name] can not be opened."
Reason: The file can not be opened.
Solution: To check whether the file exists or not.

gg. "error A033: The macro is not defined."
Reason: The macro name is not defined.
Solution: To define a new macro or change a macro name.

hh. "error A034: The expression can not be calculated."
Reason: The format of the expression is not correct.
Solution: To correct the format of the expression.

ii. "error A035: The operation [operation name] is not defined."
Reason: The operation is not defined by WICE.
Solution: To check the user menu of WICE.

jj. "error A036: The PUBLIC or EXTERN label [label name] must be address label."
Reason: The label defined by “PUBLIC” or “EXTERN” instruction must be address label.
Solution: To remove the public or extern variable.

kk. "error A037: The operand value can not be calculated."
Reason: The format of the operand expression is not correct.
Solution: To correct the expression of the operand.

ll. "error A038: The symbol [symbol name] is not extern symbol."
Reason: The symbol is not defined as internal or external.
Solution: To define the symbol.

mm. "error A039:The reference number is over [number] limitation."
Reason: The reference number in the “PUBLIC” or “EXTERN” instruction is over limitation.
Solution: To divide the line into two or more.

nn. "error A040: The length of file name [file name] exceeds 256."
Reason: The length of file name is over 256.
Solution: To change the file name or directory.

oo. " Warning A050:The symbol length is exceed 32."
Reason: The length of variable is over 32.
Solution: To change the variable.

pp. " Warning A051:The number of message is over 500 "
Reason: The number of message is over 500.
Solution: To reduce the number of message.

qq. " Warning A052:The [number] to express [number] "
Reason: Transform negative number to hexadecimal.

rr. "Warning A053:The call/jmp address is not over page. "
Reason: The target address and “call” or “jmp” instruction is in the same page.

ss. " Warning A054 :'%s' :unreferenced variable. "
Reason: The variable not use in the project.

A.3 Class L: linker error

a. **"error L001: Memory of [stack type] stack overflows."**
Reason: O.S. can not allocate extra memory.
Solution: To close unused editor window, or close unexecuted application.

b. **"error L002: The file [file name] can not be found."**
Reason: The file can not be found.
Solution: To check whether the file exists or not.

c. **"error L003: The Object file format does not belong to EMC object file format."**
Reason: The object file format does not belong to EMC object file format.
Solution: To reassemble the source file.

d. **"error L004: Symbol [symbol name] is not defined."**
Reason: The symbol is not defined.
Solution: To define the symbol.

e. **"error L005: Public symbol [symbol name] is conflicted."**
Reason: The public symbol is defined more than once.
Solution: To redefine the public symbol.

f. **"error L006: ROM address [number] is conflicted."**
Reason: The program address is conflicted with another.
Solution: To use function of "[Edit]->[Find]" menu to find the address in the list file; then to update the address of the program position.

g. **"error L007: The file [file name] can not be created."**
Reason: The file can not be created.
Solution: To check whether the disk is full or not and whether the system is stable or not.

h. **"error L008: Line [number]: The machine address [number] exceeds ROM size [number]."**
Reason: The address of program address is over the ROM size.
Solution: To reduce the program code.

i. **"error L009: No project file is active."**
Reason: No project is opened.
Solution: To open or create a project.

j. **"error L010: No output window is found."**
Reason: Output window is not opened.
Solution: To select [View]->[Output], and then an output window displays.

A.4 Class D: Debugger error

a. **"error D001: The ICE memory is error ."**
Reason: The SRAM in the ICE is error.
Solution: To change the ICE SRAM.

b. **"error D002: CDS size = [number] does not match the ROM size = [number]."**
Reason: The CDS size is not the same as the ROM size.
Solution: To check project microcontroller name whether it is the same as WICE system micro controller name or not.

c. **"error D003: The project MCU type [type name] does not match [ICE name] ICE."**
   Reason: The project micro controller name is not the same as WICE system microcontroller name.
   Solution: To create a new project or reconnect with another ICE.
d. **"error D004: The line of file does not transfer to machine address."**
   Reason: The program line is unable to assemble.
   Solution: To check the syntax of the program line.
e. **"error D005: The breakpoint group number is over 64 ."**
   Reason: The breakpoint group number is over 64.
   Solution: To remove the extra breakpoint groups.
f. **"error D006: The ICE is not connected to PC."**
   Reason: The ICE is not connected to PC.
   Solution: To check power supply, ICE crystal, connected printer port, and so on.
g. **"error D007: The printer port is not connected."**
   Reason: The printer port is not connected.
   Solution: To check power supply, connected parallel port, and so on.
h. **"error D008: The number is invalid."**
   Reason: The number is invalid.
   Solution: To rewrite the number expression.
i. **"error D009: The number of messages is over [number]."**
   Reason: The number of messages is over limit in output window.
   Solution: To reduce the message is defined by user, example for “MESSAGE” instruction.
j. **"Warning D010: The address [number] does not match the source file."**
   Reason: The program address does not match the source file line.
   Solution: To add the program source of the address, or to check whether the crystal is normal or not.
k. **"Warning D011: Memory checked is error."**
   Reason: The ICE SRAM memory is error.
   Solution: To change the ICE SRAM, or to check whether the crystal is normal or not.
l. **"error D012: Syntax error."**
   Reason: The syntax of source file has a severe error.
   Solution: To correct the source file.
m. **"error D013: Memory address [number] is error !"**
   Reason: The address of ICE SRAM is error.
   Solution: To change the ICE memory.
n. **"error D014: Can't find the breakpoint address of [program line]."**
   Reason: The breakpoint line does not have program address.
   Solution: To redefine the breakpoint.
o. **"error D015: The number of symbols is over [number]."**
   Reason: Too many symbols are defined.
   Solution:. To divide the program into two or more small programs
p. **" error D016: Please dump program before adding label to watch. "**
   Reason: To dump program before adding label to watch.
   Solution: To dump program to ICE.
q. **" error D017: Trace log is empty. "**
   Reason: No more data is found in trace log.

Solution: Can not execute the operation.

r. " error D018: No trace item in trace log. "
Reason: No more trace item is found in trace log.
Solution: Can not execute the operation.

Appendix B
C Conversion Table

B-1 Conversion between C and Assembly Codes

The assembly code was generated by the WicePlus.

Description	C Statement Example	Assembly Code	Conversion Rate (Compiler's Code Size / General User's Code Size * 100%)
Integer Variable	intVar1 = 0xFF;	MOV A, @0xFF MOV %intVar1, A	100% (2 / 2 * 100)
	intVar2 = intVar1;	MOV A, %intVar1 MOV %intVar2, A	100% (2 / 2 * 100)
Character Variable	charVar1 = 0xFF;	MOV A, @0xff MOV %charVar1, A	100% (2 / 2 * 100)
	charVar2 = intVar1;	MOV A, %charVar1 MOV %charVar2, A	100% (2 / 2 * 100)

Short Variable	shortVar1 = 0x1234;	MOV A, @0x34
		MOV %shortVar1, A
		MOV a, @0x12
		MOV %shortVar1+1, A
			100% (4 / 4 * 100)

	shortVar2 = shortVar1;	MOV A, %shortVar1
		MOV %shortVar2, A
		MOV A, %shortVar1+1
		MOV %shortVar2+1, A
			100% (4 / 4 * 100)

Long Variable	longVar1 = 0x123456;	MOV A, @0x56
		MOV %longVar1, A
		MOV A, @0x34
		MOV %longVar1+1, A
		MOV A, @0x12
		MOV %longVar1+2, A
			100% (6 / 6 * 100)

	longVar2 = longVar1	MOV A, %longVar1
		MOV %longVar2, A
		MOV A, %longVar1+1
		MOV %longVar2+1, A
		MOV A, %longVar1+2
		MOV %longVar2+2, A
			100% (6 / 6 * 100)

For loop	for (i = 0; i < 5; i++)
	{
	……
	}
		CLR %i
		JMP L2
		L1:
		……
		L2:
		INC %i
		MOV A, @0x05
		SUB A, 0x14
		JBS 0x03, 0
		JMP L1
			100% (7 / 7 * 100)

While statement	while (cnt != 1)
	{
	……
	}
		L1:
		…
		
		MOV A, %cnt
		XOR A, @0X01
		JBS 0X03,2
		JMP L1
			100% (4 / 4 * 100)

Do-while statement	do
	{
	……
	} while (cnt != 1);
		L1:
		……
		MOV A, %cnt
		MOV A, @0x01
		XOR A, @0x01
		JBS 0x03, 2
		JMP L1
			100% (4 / 4 * 100)

Do-while statement	do { Var_c2++; }while(-- var_c1);	L1: INC %var_c2; DJZ %var_c1; JMP L1	100%(3/3*100)

If-else statement	unsigned int cnt;
	if (cnt == 0)
	{
	……
	}
	else if (cnt < 5)
	{
	……
	}
	else
	{
	……
	}
		MOV A, %cnt
		JBS 0x03, 2
		JMP L1
		……
		JMP ENDIF
		L1:
		MOV A,@0X05
		SUB A, %cnt
		JBC 0x03, 0
		JMP ENDIF
		……
		JMP L2
		L2:
		……
		ENDIF:
			100% (10 / 10 * 100)

Switch statement	unsigned int cnt;
	switch(cnt)
	{
	  case 1:
	  ……
	  break;
	  case 2:
	  ……
	  break;
	  case 3:
	  ……
	  break;
	  default:
	  ……
	  break;
	}
		MOV A,%cnt
		MOV 0X14,A
		MOV A,0X14
		XOR A,@0x01
		JBC 0X03,2
		JMP case 2
		MOV A,0X14
		XOR A,@0X02
		JBC 0X03,2
		JMP case 2
		MOV A,0X14
		XOR A,@0X3
		JBC 0X03,2
		JMP case 3
		JMP default
		
		Case 1:
		JMP …ENDSWITCH
		Case 2:
		JMP ENDSWITCH
		Case 3:
		JMP ENDSWITCH
		Case 4:
		default
		
		ENDSWITCH
			106% (18 / 17 * 100)

Function	main()
	{
	  int i;
	  i = fun(3);
	
	  return;
	}
	
	int fun(int in)
	{
	  return in+1;
	}
		; using EM78806B
		MOV A, @0x03
		BANK @0
		MOV %in, A
		CALL FUN
		MOV A, 0x10
		BANK @0
		MOV %i, A
		RET
		
		FUN:
		MOV A, 0x14
		BANK @0
		MOV %temp1, A
		MOV A,%in
		MOV 0x14,A
		MOV 0X10,A
		MOV A,@0X01
		ADD 0X10, A
		MOV A,%temp1
		MOV 0X14,A
		RET
			136% (19 / 14 * 100)

Const array	const int myConst[5] = {1, 2, 3, 4, 5};
	
	main()
	{
	int i;
	
	i = myConst[3];
	
	return;
	}
		; using EM78569
		
		MOV A, @0x3D
		MOV 0x19, A
		MOV A, @0x1F
		MOV 0x1A, A
		PAGE @0x0F
		CALL 0x280
		PAGE @0x00
		BC 0x04, 6
		BC 0x04, 7
		MOV 0x20, A
		…
		BC 0X03,0
		RLCA 0x1A
		TBL
		…
		PAGE @0x0F
		JMP 0x2FE
		MOV A, 0x19
		TBL
		….
		
		RETL @0x01
		RETL @0x02
		RETL @0x03
		RETL @0x04
		RETL @0x05
			162% (21 / 13 * 100)

Register page	unsigned int myR5P0 @0x05: rpage 0;
	unsigned int myR5P1 @0x05: rpage 1;
	unsigned int myR5P2 @0x05: rpage 2;
	
	myR5P0 = 0x12;
	myR5P1 = 0x34;
	myR5P2 = 0x56;
		; using EM78P468N
		MOV A, @0x12
		BS 0X03,6
		MOV 0x05, A
		
		MOV A, @0x34
		BS 0x03, 6
		BC 0x03, 7
		MOV 0x05, A
		
		MOV A, @0x56
		BC 0x03, 6
		BS 0x03, 7
		MOV 0x05, A
			100% (11 / 11 * 100)

I/O control page	io unsigned int myIO6P0 @0x06: rpage 0;
	io unsigned int myIO6P1 @0x06: rpage 1;
	io unsigned int myIO7P1 @0x07: rpage 1;
	
	myIO6P0 = 0x00;
	myIO6P1 = 0xFF;
	myIO7P1 = 0x55;
		; using EM78569
		MOV A, @0x00
		BC 0x03, 5
		IOW 0x6
		
		MOV A, @0xFF
		BS 0x03, 5
		IOW 0x6
		
		MOV A, @0x55
		IOW 0x7
			100% (8 / 8 * 100)

RAM bank	unsigned int myData1 @0x20: bank 0;
	unsigned int myData2 @0x21: bank 0;
	unsigned int myData3 @0x21: bank 1;
	
	myData1 = 1;
	myData2 = 2;
	myData3 = 3;
		; using EM78569
		MOV A, @0x01
		BC 0x04, 6
		BC 0x04, 7
		MOV 0x20, A
		MOV A, @0x02
		MOV 0x21, A
		MOV A, @0x03
		BS 0x04, 6
		BC 0x04, 7
		MOV 0x21, A
			100% (10 / 10 * 100)

Bit data type	bit myB0R6P0 @0x06@0x00: rpage 0;
	bit myB2R6P0 @0x06@0x02: rpage 0;
	
	myB0R6P0 = 1;
	myB2R6P0 = myB0R6P0;
		
		BS 0x06, 0
		BC 0x06, 2
		JBC 0x06, 0
		BS 0x06, 2
			133% (4 / 3 * 100)

Indirect addressing	indir unsigned myData1 @0x30: ind 0;
	indir unsigned myData2 @0x05: ind 1;
	
	myData1 = 0x55;
	myData2 = 0xAA;
		; using EM78806B
		MOV A, @0x55
		MOV 0x1B, A
		MOV A, @0x30
		MOV 0x18, A
		MOV A, @0x00
		MOV 0x19, A
		MOV A, @0x00
		MOV 0x1A, A
		MOV A, 0x1B
		CALL INDIR
		MOV A, @0xAA
		MOV 0x1B, A
		MOV A, @0x05
		MOV 0x18, A
		MOV A, @0x00
		MOV 0x19, A
		MOV A, @0x01
		MOV 0x1A, A
		MOV A, 0x1B
		CALL INDIR
		……
			146% (35 / 24 * 100)

		INDIR:
		BC 0x05, 0
		MOV 0x1B, A
		MOV A, 0x1A
		JBS 0x03, 2
		JMP 0x081
		MOV A, 0x18
		IOW 0x9
		MOV A, 0x1B
		IOW 0xA
		RET
		
		LCDRAM:
		MOV A, 0x18
		MOV 0x0A, A
		MOV A, 0x1B
		MOV 0x0B, A
		RET	

Bitwise operation
(all variables are “unsigned int” data type)
	f = e & d;
	(f = e ^ d;)
	(f = e | d;)
		MOV A, %e
		AND A, %d
		(XOR A, %d)
		(OR A, %d)
		MOV %f, A
			100% (3 / 3 * 100)

	f=~e;	COMA %e MOV %f, A	100%(2/2*100)
	f &=e; (f ^=e;) (f |= e)	MOV A, %e AND %f , A (XOR %f, A) (OR %f, A)	100%(2/2*100)
	f = e >> 1;	BC 0x03, 0 RRCA %e MOV %f, A	100% (3 / 3 * 100)
	f = e << 1;	BC 0x03, 0 RLCA %e MOV %f, A	100% (3 / 3*100)
	f>>=3;	BC 0x03, 0 RRC %f BC 0x03, 0 RRC %f BC 0x03, 0 RRC %f	100%(6/6*100)
	f<<=3	BC 0x03, 0 RLC %f BC 0x03, 0 RLC %f BC 0x03, 0 RLC %f	100%(6/6*100)
	f>>=4	SWAPA 0x06 AND A, @0x0F MOV 0x06, A	100%(3/3*100)
	f<<=4	SWAPA 0x06 AND A, @0xF0 MOV 0x06, A	100%(3/3*100)

	f>>=6;	SWAP 0x06 RRC 0x06 RRCA 0x06 AND A, @0x03 MOV 0x06, A	100%(5/5*100)
	f<<=6;	SWAP 0x06 RLC 0x06 RLCA 0x06 AND A, @0xC0 MOV 0x06, A	100%(5/5*100)

	f=(e<<5) | d;	MOV A, %e
		MOV 0x14, A
		SWAP 0x14
		RLCA 0x14
		AND A, @0xE0
		OR A, %d
		MOV %f, A
			100%(7/7*100)

	f=(f & const.1) | const. 2	MOV A, 0x06
		AND A, const. 1
		OR A, const. 2
		MOV 0x06, A
			100%(4/4*100)

Arithmetic expression
(all variables are “int” data type)
	f = e + d;	MOV A, %e
		ADD A, %d
		MOV %f, A
			100% (3 / 3 * 100)

	f = e – d;	MOV A, %d
		SUB A, %e
		MOV %f, A
			100%(3/3*100)

	f++;	INC %f	100% (1 / 1 * 100%)
	f—;	DEC %f	100% (1 / 1 * 100%)

	c = a * b;	MOV A, %a
		MOV 0X1C,A
		MOV A, %b
		MOV 0X18, A
		CLRA
		L1:
		ADD A, 0X1C
		DJZ 0X18
		JMP L1
		MOV %c, A
			100%(9/9*100%)

	c = a / b;	MOV A, %a
		MOV 0x1C, A
		MOV A, %b
		CLR 0x18
		L1:
		SUB 0x1C, A
		JBC 0x03, 0
		INC 0x18
		JBC 0x03, 0
		JMP 0x3BB
		MOV A, 0x18
		MOV %c, A
			100%(11/11*100%)

Compound assignment
(all variables are “int” data type)
	f += e;
	(f -= e;)
	(f &= e)
	(f ^= e)
	(f |= e)
		MOV A, %e
		ADD %f, A
		(SUB %f, A)
		(AND %f, A)
		(XOR %f, A)
		(OR %f, A)
			100% (2 / 2 * 100%)

	f >>= 1;	BC 0x03,0 RRC %f	100% (2 / 2 * 100%)
	f <<= 1	BC 0x03,0 RLC %f	100% (2 / 2 * 100%)

Appendix C
Frequently Asked Questions about Assembly (FAQ)

C.1 Q：ICE can not connect to PC?

A：Please check the following steps:

Step 1：Check the power supply with output around the range of 17~23 voltages.
Step 2：Check the printer cable between ICE and PC.
Step 3：Check the oscillator on the hardware.
Step 4：From the Option menu, choose ICE code option. Check the type of oscillator in the code option.
Step 5：If the problem still exists, then check again with another PC.
Step 6：Repeat from Step 1.
Step 7：If it is still not solved, please call EMC.

C.2 Q：The connection from ICE to PC is successful, but program dumping fails.

A：Please check the following steps:

Step 1： From the Option menu, choose ICE code option. To check the type of oscillator in the code option.
Step 2： Please check whether the type of microcontroller is the same as that in hardware.

C.3 Q：If ICE memory is checked, and the result of ICE memory is error.

A：Please check the following steps:

Step 1： From the Option menu, choose ICE code option. To check the type of oscillator in the code option.
Step 2： Please check whether the type of microcontroller in software is the same as that in hardware.
Step 3： Please change ICE SRAM.

C.4 Q：If users try to reconnect the same ICE from PC, why does the execution become slower and slower ?

A： The timing of communication between ICE and PC is stretched if the connection is tried again and again.

C.5 Q：Why can’t “Step Into” be executed?

A： Please check whether the type of microcontroller in software is the same as that in hardware.

C.6 Q：If source files are located in Novell file server, why is the program reassembled again at each execution time?

A： It is because of the time difference between PC and server. Therefore, it is better to locate all source files in one computer no matter that is network computer or local one.

Appendix D
Frequently Asked Questions About Tiny C Compiler (FAQ)

D.1 Q:What is the maximum number of the function parameters?

A: It depends on the RAM bank size (about 32 or 31 bytes).

D.2 Q:In a function, what is the maximum depth of the function call?

A: It depends on the hardware stack depth or size.

D.3 Q:What is the maximum array dimension as well as maximum array element?

A: It depends on the RAM bank size (about 32 or 31 bytes).

D.4 Q:Is there any error message when the code exceeds the ROM size?

A: Yes, the linker will report an allocation error.

D.5 Q:In a high level interrupt subroutine, can user allocate the address in the ROM? (e.g., using “page” data type, putting “_asm{ org xxx}” before a subroutine, etc.)

A: No! This may cause unpredictable error.

D.6 Q:Is “static” used in the same way as in ANSI C?

A: Yes.

D.7 Q:Is there any error message in case user defines too many variables in the “const” that exceeds the ROM space?

A: Yes, the linker will report an allocation error.

D.8 Q:How do I declare the variable in *.h file and using not only in one .c file?

A: for example, declare in *.h file like that:
extern io unsigned int DIRPORT6;
and you have to write like below just only one *.c file like that:
io unsigned int DIRPORT6 @0x06: iopage 0;

D.9 Q: Should I change any program page or bank?

A: If you just develop your program in C language, you don’t have to change any program page, register page and ram bank, and so on. But If you use inline assembly in your program, you have to save and restore about page or bank.

D.10 Q: May I know how many stacks I have called?

A: Yes. In C developed environment, after compiling, user can know how many function call depth in Information, Output Window.

D.11 Q: Does C compiler just occupy 0x10~0x1F general purpose ram?

A: Well, almost C compiler just occupies 0x10~0x1F general purpose register. But If there are some arguments in call functions, compiler will use some others ram in 0x20~0x3F, bank 0 ~ bank 3. So, we suggest users use global variables to replace arguments in call function.

Users always have to note that there are some ram spaces used in interrupt save procedure and interrupt service procedure. If you don't use these ram space again.

D.12 Q: Does C compiler support this item, such as EM78P510N?

A: TCC2 supports all EM78 series IC except EM78x680, EM78x611.

D.13 Q: How do I use macro with variable?

A: for example:

```
io unsigned int P6CR@0x06:iopage 0;
…
#define set_output(port, bit)   _asm{ ior port};\
                                _asm{ and a,@(~(1<<bit))};\
                                    _asm{ iow port}

…
set_output(%P6CR,0x04);
```

D.14 Q: Does C Compiler support all assembly instruction?

A: TCC2 supports all assembly instruction. But some instructions have to implement by

the form of inline assembly. Such as MUL, TBRD, TBWD, and MOV R, R.

D.15 Q: How can clear all ram in all banks?

A: There are two key points in this issue. One is to keep bit 5, R4, as 1. The other is to switch bank correctly. But some IC uses a non-global register to switch ram bank, such as EM78P510N. So users have to know the peculiar to avoid fault. For example:

```
unsigned int iR0:0x0:rpage 0;
unsigned int RSR:0x4:rpage 0;
unsigned int BRSR:0x5:rpage 0;
unsigned int i:0x11:rpage 0;
…
for(i=0;i<0x8;i++)
{
  BRSR=i;
  For(RSR=0xFF;RSR>=0xE0;RSR--)
  {
    iR0=0;
     }
}
```

Appendix E
The IC option comparative table

After the WicePlus model reduces staff the IC option comparative table.

Contains WicePlus2.2xxxx is new IC option, before is the old IC option.

New IC option	Old IC option
EM78P132	EM78P155N
EM78P210N	EM78P211N
	EM78P212N
EM78P220N	EM78P221N
	EM78P222N
EM78P259N	EM78P258N
	EM78P259N
	EM78P260N
EM78P142	EM78P340N
EM78P342N	EM78P341N
	EM78P342N
	EM78P343N
EM78P346N	EM78P345N
	EM78P346N
	EM78P347N
EM78P418N	EM78P417N
	EM78P418N
	EM78P419N
EM785840 or EM78P5840N	EM785840
	EM785841